NOTE TO USERS

The diskette is not included in this original manuscript. It is available for consultation at the author's graduate school library.

This reproduction is the best copy available.

DEVELOPMENT, ANALYSIS AND COMPARISON OF CONNECTIONIST MODELS FOR REAL TIME OPTIMISATION

by

Khairiyah Mohd-Yusof

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Chemical Engineering

Waterloo, Ontario, Canada, 2001

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

ABSTRACT

This research investigates the development of different types of artificial neural networks (ANN) and grey-box ANN models for real time optimisation (RTO) of chemical processes. Therefore, must be easily computed, stable, and easily updated and maintained. Due to this, the models have a predominantly connectionist structure. All the models were developed and simulated under MATLAB environment.

Initial investigations were focused on a methanol-water flash system, which is simple, yet realistic in representing the non-linearity of multivariable chemical processes. Following this, ANN models were then developed for a crude oil distillation column, which is a more complex industrial process. Training and testing data for the network models were generated using steady state process models simulated in the Aspen Plus steady state process simulator. The sensitivity analysis feature in Aspen Plus was utilised to generate a large amount of data in a single simulation run.

Three standard ANN models were developed for the M-W flash system: multi-layer perceptrons (MLP) using backpropagation training with variable learning rate, MLP using Levenberg-Marquadt, and radial basis function networks (RBFN). Of the three standard ANN models, the RBFN was found to give the best result, and was thus selected as the base case for comparison with other models.

The RBFN models were able to model the M-W flash system well, except for y, the composition of methanol in the vapour outlet stream. Different combinations of output variables affect the predictions of the model. In general, grouping suitable output variables combinations in a network model gave better predictions.

More complex models were required for better prediction of y because of the discontinuity in y that exists in the change between the single-phase and the two-phase

region. Three groups of models were developed: hierarchically structured neural network (HSNN), serial network models and hybrid ANN-first principles models (FPM). The models in all the three groups managed to improve the prediction of the base model.

Among all the models, the output-tuned HSNN model that was designed in series with a serial RBFN model provided the best prediction of y. Moreover, the structure of the output-tuned HSNN provided means for incorporating constraints into the network. The constraints used are simple and readily known information. The completely driven HSNN, was also able to perform as well, when there is an input variable that directly and strongly influenced y. In both types of HSNN, prior knowledge embedded into the network structure allowed for significant improvements over the base case.

The results of the hybrid ANN-FPM also showed a significant improvement over the base case in predicting y. Performance of the hybrid model, however, depended on the availability of accurate parameters needed by the FPM. Therefore, a hybrid ANN-FPM would require extensive prior information.

For the crude oil distillation tower, a standard RBFN was able to provide a highly satisfactory model. Proper grouping of related variables not only improved predictions, but also allowed for the complex, multivariable model to be more manageable while avoiding the "curse of dimensionality". The RBFN model for one of the crude tower sections was also able to give good predictions when tested for range and dimensional extrapolation. Since standard RBFN gave sufficiently accurate predictions, developing more complex models was deemed to be unnecessary.

This study showed that ANN and grey box ANN models have the potential to model chemical processes for RTO. Various methods exist to take advantage of readily available information that can be utilised to overcome modelling difficulties. Most significantly, since these models can be easily developed and updated, they are suitable for practical industrial applications.

ACKNOWLEDGEMENTS

First and foremost, I thank God Almighty for giving me the strength to complete my research. I would also like to thank the following people who assisted me in my research:

- Prof. Peter Douglas and Prof. Fakhri Karray for their advice and encouragement.
- Assoc. Prof. Dr. Hishamuddin Jamaluddin of Universiti Teknologi Malaysia for his assistance in helping me understand the HSNN algorithm.
- Engineers and plant personnel at PETRONAS Penapisan Melaka refinery for helping me learn about the crude distillation tower.

I would also like to thank my parents, Norashikin and Mohd. Yusof, for their constant support and prayers. Last but not least, I would like to thank my husband, Helmi, and my children, Ruqayyah, Sofwan and Muhammad, for their love, prayers, sacrifice and encouragement. May God protect and guide all of you.

CONTENTS

ABSTRACT iv

ACKNOWLEDGEMENTS vi

LIST OF TABLES xii

LIST OF FIGURES xiii

1. INTRODUCTION 1

1.1 Background 1

1.2 ANN in Chemical Engineering 4

1.3 Objective 7

1.4 Significance of Research 9

1.5 Thesis Outline 10

2. LITERATURE SURVEY 12

2.1 Introduction 12

2.2 Real Time Optimization (RTO) 13

2.2.1 Introduction 13

2.2.2 Previous Research 16

2.3 Artificial Neural Networks 17

2.3.1 Introduction to Artificial Neural Networks 18

2.3.2 Network Topologies 21

2.3.3 Developing ANN Models 22

2.3.4 Multilayer Perceptrons 25

2.3.5 Radial Basis Function Network (RBFN) 26
2.4 Hybrid FPM-ANN Models 29
2.4.1 Background 29
2.4.2 Structures of Semi-mechanistic Models 30
2.4.3 Other Forms of Hybrid Models 33
2.4.4 FPM for Training ANN 34
2.5 Modular Neural Networks 35
2.5.1 Background 35
2.5.2 Classes of Modular Neural Networks 36
2.6 Summary 42

3. PROCESS DESCRIPTIONS 44
3.1 Introduction 44
3.2 Simple Process, The M-W Flash 45
3.3 Industrial Process 49
3.4 Data Generation 53
3.5 Summary 57

4. DEVELOPMENT OF CONNECTIONIST MODELS 58
4.1 Introduction 58
4.2 Multi-layer Perceptrons (MLP) 59
4.3 Radial Basis Function Networks (RBFN) 60
4.4 Hierarchically Structured Neural Networks (HSNN) 60
4.4.1 Linear-nonlinear HSNN 61
4.4.2 Nonlinear-nonlinear HSNN 65
4.4.3 Output-tuned HSNN 68
4.5 Serial Network Models 71
4.6 Hybrid Network Models 73

4.7 Summary 76

5. RESULTS AND DISCUSSIONS 77

5.1 Overview 77

5.2 Standard ANN 80

5.2.1 Base Case Model Selection 80

5.2.2 RBFN Models for M-W System 81

5.2.3 RBFN Models for B-T System 82

5.2.4 RBFN Models for Two-Phase Region 84

5.2.5 Predicting y 86

5.3 Completely Driven HSNN 87

5.3.1 Linear-nonlinear HSNN 87

5.3.2 Nonlinear-nonlinear HSNN 90

5.4 Serial Network Models 91

5.4.1 Serial RBFN-RBFN Models 91

5.4.2 Serial RBFN-RBFN - Output-tuned HSNN 92

5.5 Hybrid Network Models 94

5.5.1 Hybrid RBFN-FPM 94

5.5.2 Hybrid RBFN-FPM-RBFN 95

5.6 Comparison of the Prediction of y 96

5.7 Crude Distillation Tower 107

5.7.1 Sections of the Crude Tower 107

5.7.2 Comparison between RBFN and MLP 108

5.7.3 Grouping of Variables 108

5.7.4 Overall Prediction 110

5.7.5 Simple Range and Dimensional Extrapolation 111

5.7.6 Objective Function for RTO 112

5.8 Summary 114

6. CONCLUSIONS 117
6.1 Conclusions 117
6.2 Contributions 118
6.3 Recommendations 119

REFERENCES 120

A Background on Neural Networks 133
A.1 Neurons and Neural Networks 133
A.2 Developing ANN Models 135
A.3 Multilayer Perceptrons 137
A.4 Radial Basis Function Network (RBFN) 141

B Update Equations Derivation for HSNN 147
B.1 Update Equations for Linear-Nonlinear HSNN 147
B.2 Update Equations for Nonlinear-nonlinear HSNN 149
B.3 Update Equations for Output-tuned HSNN 155

C Training and Testing Data 161
C.1 Training and Testing Data for the Flash Systems 161
C.2 Training and Testing Files for the Crude Distillation Tower 176

D MATLAB Programs 178
D.1 Sample Program for Standard RBFN 178
D.2 Sample Program for MLP 180
D.3 Sample Program for Hybrid RBF-FPM-RBF 183
D.4 Sample Program for Linear-nonlinear HSNN 188

D.5 Sample Program for Nonlinear-nonlinear HSNN 191
D.6 Sample Program for Output-Tuned HSNN 194

LIST OF TABLES

3.1 Product specifications and manipulated variables of the crude tower ... 53

3.2 Input and output variables for each section of the crude distillation column. 56

5.1 Prediction of V using 150 and 300 training data points 79

5.2 Comparison of the best results obtained with RBFN and two MLP models. 81

5.3 Results of RBF network models 82

5.4 Results of the B-T system using RBFN models 84

5.5 Results of standard RBFN model for M-W system in the two-phase region. 85

5.6 Results of linear-nonlinear HSNN models 89

5.7 Results of nonlinear-nonlinear HSNN models 90

5.8 Results of serial RBF network models 92

5.9 Results of serial RBFN - RBFN - output-tuned HSNN models 93

5.10 Results of Type 1 hybrid structure 95

5.11 Results of hybrid structure 96

5.12 RMS errors for prediction of y using the different models 97

5.13 Overall results for the top section of the main crude distillation column using RBFN and feedforward network with BP. 108

5.14 RMS errors of variables in top and HN sections of the crude tower. 109

5.15 Overall results for all sections in the crude distillation tower 110

5.16 RMS errors for range and dimensional extrapolation 112

LIST OF FIGURES

2.1	General configuration of a real time optimiser	15
2.2	Schematic diagram of a three-layer feedforward network	22
2.3	Schematic diagram of a recurrent network	23
2.4	General structure of a radial basis function network	27
2.5	Serial ANN-FP hybrid model	30
2.6	Parallel ANN-FP hybrid model	32
2.7	Combination of parallel and serial ANN-FPM hybrid model	32
2.8	An alternative form of serial hybrid FP-ANN model	33
2.9	Structure of hierarchical network from Chang and Mavrovouniotis [1992].	38
2.10	Mixture of experts partitioned by gating network as proposed by Jordan and Jacobs [1995].	39
2.11	Hierarchically structured master and slave networks	40
2.12	Structure for stacked networks	42
3.1	Schematic diagram of a general two-component flash column	46
3.2	A general two-component phase diagram	47
3.3	Crude tower flow diagram	50
4.1	General structure of linear-nonlinear HSNN	62
4.2	Structure of linear-nonlinear HSNN for M-W system	61
4.3	A nonlinear-nonlinear HSNN	66
4.4	Output-tuned HSNN	69
4.5	Serial network model	71
4.6	Serial RBFN - RBFN - output-tuned HSNN	72
4.7	Type 1 serial hybrid model with K-value (Model 1)	73
4.8	Type 1 serial hybrid model with component balance (Model 2)	75
4.9	Type 2 hybrid model (Model 2)	75

4.10	Type 2 hybrid model (Model 3)	75
4.11	Type 2 hybrid model (Model 4)	76
5.1	Plot of normalised actual V versus normalised predicted V	86
5.2	Plot of y predicted using standard RBFN (base case) ..... (RMS = 0.0119).	98
5.3	Plot of y in the two-phase envelope only (RMS = 0.0065)	98
5.4	Plot of y predicted using linear-nonlinear HSNN (F slave) ..... (RMS = 0.0885).	100
5.5	Plot of y predicted using linear-nonlinear HSNN ..... (K-values with errors as slave input) (RMS = 0.0857).	100
5.6	Plot of y predicted using linear-nonlinear HSNN ..... (K-values with errors as slave input) (RMS = 0.0864).	101
5.7	Plot of y predicted using linear-nonlinear HSNN (K-values ..... without errors as slave input) (RMS = 0.0344).	101
5.8	Plot of y predicted using nonlinear-nonlinear HSNN (z slave) ..... (RMS = 0.0820).	102
5.9	Plot of y predicted using serial RBFN-RBFN (RMS = 0.1031)	104
5.10	Plot of y predicted using serial RBFN - RBFN - ..... output-tuned HSNN (FPM) (RMS = 0.0074).	104
5.11	Plot of y predicted using hybrid RBFN - FPM (RMS = 0.0346)	106
5.12	Plot of y predicted using hybrid RBF-FPM-RBF (with ..... K-values without errors as FPM input) (RMS = 0.0202).	106
A.1	Schematic diagram of a neuron	135
A.2	General structure of a radial basis function network	142
A.3	Gaussian basis function	143

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Real time optimisation (RTO), or on-line optimisation, is the continuous evaluation and adjustment of process operating conditions to maximise economic productivity subject to constraints. It is generating much interest in industry because of the increasingly competitive environment of the current global economy. Traditionally, execution of RTO requires rigorous plant models.

Good process models are critical for a successful implementation of RTO. Currently, most RTO implementation uses rigorous first principles mathematical models, which are not only complex and costly to develop, but also cumbersome and difficult to maintain because of the skill and time requirements [Naysmith, 1997]. The high computation time required in solving the models is also a major problem for on-line applications. In fact, developing reliable models for a chemical process is a major obstacle in implementing advanced control and optimisation because of the complexity and cost involved [Hussain, 1999]. This leads to the quest for finding other types of suitable models, such as off-the-shelf commercial simulation packages or purely black box models like artificial neural networks (ANNs).

In a previous work, Naysmith [1997] successfully developed and executed complete RTO loops on stabilizer-splitter distillation columns using two commercial simulation packages, Aspen Plus and Speedup. Both packages use rigorous first

principles (FP) mathematical models to simulate chemical processes. Nevertheless, Naysmith found out that models developed using both packages were not practical for industrial use because of large computation times, convergence difficulties and failures. For example, in Naysmith's work, the total CPU time in minutes taken for data reconciliation and economic optimisation using an Aspen Plus model for a stabiliser-splitter process in a refinery calculated on IBM RS 6000 Model 530H computer was 506.3 minutes.

FP based models, or white box models, requires in-depth knowledge of material, energy and momentum conservation, as well as thermodynamics and kinetics, of the process. These models are also called mechanistic or physical models. A complete chemical process model can result in thousands of equations. As a consequence, the model is complex and requires a high level of expertise to develop and maintain. Solving these equations is equally difficult and computationally time-consuming. Thus, both model development and maintenance are expensive. Nevertheless, good FP models can extrapolate well and are extremely useful for understanding and analysing processes. There are commercial packages available to assist in modelling and simulation of chemical operations or plants. These packages are much easier to use than building a model from scratch. Moreover, most of the steady-state simulation packages also have good graphical user interface. Unfortunately, computation using these packages is slow, making on-line implementation impractical.

An alternate approach to model a process is to use artificial neural networks (ANN). ANNs have generated much interest in the chemical engineering community for more than a decade and many industrial applications have since been reported [Nascimento, et al., 2000; Gontarski, et al., 2000; Elkamel, et al., 1999; Meghlaoui, et al., 1998; Turner, et al., 1996; Baratti, et al., 1995; Cheung, et al., 1992; Thibault and Grandjean, 1991]. Thompson et al. [1996] and Thompson and Kramer [1994] suggested the use of ANN as the process model for RTO. In another work, an

important variable was estimated using an ANN model as part of a large rigorous model for use in on-line supervisory optimisation [Sabharwal, 1997].

ANN, a connectionist-based black box model, consists of layers of nodes with non-linear basis functions and weighted connections that link the nodes. Using the nodes and weights, the inputs are mapped to the outputs after being trained with a set of data known as training data. Multilayer feedforward ANNs have been mathematically proven to be a universal approximator [Hornik, et al., 1989]. However, since ANNs are data driven, the resulting model can only be as good as the data provided to the network for testing and training. Therefore, unlike FP models, ANNs are poor extrapolators.

A third approach of modelling pertains to construct a mixture of white box and black box models. A hybrid model involving white box and black box models is used to overcome the weaknesses while utilising the strength of both approaches. While most published works defined the combination of FP and ANN models as hybrid models, te Braake, et al. [1998] classified them as semi-mechanistic models and grey box models. The starting point of a semi-mechanistic model is a white box model, with certain parameters or variables calculated from black box models or empirical correlations. The starting point of a grey box model is a black box model, with prior knowledge being used to provide additional information.

Grey box models can also be defined to include models with structures designed to incorporate prior knowledge. Several methods of integration that were successful to certain applications had been proposed, but there were no specific proven rules. Some of the methods use a direct hybrid of ANN and FP models. Modular neural networks (MNN), which represent another class of ANN, break down large models into sub-systems that cooperate with each other to come up with the model outputs [Gallinari, 1995]. The architecture of MNN is usually designed based on prior knowledge of the process. Another class of network, Hierarchically Structured Neural Networks (HSNN) [Bittanti and Saveresi, 1998], uses prior knowledge of the output variable behaviour to divide the input variables into "master" and "slave" units. The outputs of the master

network are parameters for the slave network. Based on this, the slave network calculates the predicted output variables of the system. In this work, grey box ANN models will be used in a generic sense to cover hybrid ANN-FP models, MNN models, HSNN models and combinations of the models.

1.2 ARTIFICIAL NEURAL NETWORKS IN CHEMICAL ENGINEERING

Artificial neural networks (ANNs) are widely applied in chemical industries, especially in the area of fault diagnosis, process modelling, process control and process optimisation. Applications in chemical engineering include those in petroleum refineries [Elkamel, et al., 1999; Cheung, et. al., 1992; Thompson et. al, 1996; Zhao et. al., 1997], chemical plants [Turner, et. al., 1996; Baratti, et. al., 1995; MacMurray and Himmelblau, 1995; Bulsari, et. al., 1994], polymerisation processes [Nascimento, et al., 2000; Zhang et. al, 1995], microelectronics fabrication [Fakhr-Eddine, 1996], biotechnology [Thibault, et. al, 2000; Latrille, et. al., 1994; Schubert, et. al., 1994], metallurgical processes [Meghlaoui, et. al, 1998; Reuter, et. al., 1993], wastewater treatment [Gontarski, et al., 2000; Syu and Chen, 1998], and oil recovery [Elkamel, 1998; Elkamel, et. al, 1996].

There are several advantages of using ANN models in chemical engineering. These include:

1. ANN models are simpler to develop than FPM because a detailed knowledge of the process is not required. This directly translates into savings in time and money.
2. ANN models can be easily used, updated and maintained, making them the preferable form of models for plant engineers.

3. A large amount of data may be required to train the network. However, this is not a big problem as most chemical plant engineers are "data rich and information poor" [Venkatasubramanian and McAvoy, 1992].
4. Once trained, execution of ANN based models are very fast, even though large networks may take longer training times. ANN models are thus particularly suitable for use in on-line applications that repeatedly evaluates the model, like in RTO.

While ANN models have gained many proponents among engineers in the chemical industries, there are also some that are sceptical of the ability of ANN because of their structural weaknesses. The disadvantages of ANNs include:

1. To develop ANN models, there is no guidance for picking suitable structures and training algorithms.
2. Large number of parameters that may be used in ANN models makes it easy to overfit the data, causing poor generalisation. ANN models are also known to be poor extrapolators.
3. Training algorithms that use steepest descent methods to compute the connection weights tend to be trapped in local minima. This results in inconsistent solutions that are highly dependent on the initial values, which is especially apparent in large networks that are very common in chemical engineering.
4. Incorporating prior knowledge into an ANN is also difficult. Once trained, ANN models do not carry physical significance and therefore cannot yield any insight to the process.
5. ANN models are poor extrapolators. They are usually only reliable within the range of data that they had been trained for.

Lately, there have been several studies that reported a preference using ANN models even though rigorous FP models for the processes are available [Nascimento, et al., 2000; Altissimi, et al., 1998]. The FP models were used to generate data to develop

ANN models. ANN models are especially suitable for on-line applications because of the relatively short computational time to solve the model and its ability to accurately represent the model. In optimisation, this is especially advantageous since the model must be accessed by the optimiser and computed repeatedly. Altissimi, et al. [1998] successfully applied ANN models of a gas separation unit in RTO. The ANN model was developed from data generated by the plant model simulated in Aspen Plus. Successive quadratic programming (SQP) was then used to optimise the profit of the process unit. Replacement of the rigorous FP model with the ANN model reduced computation time by at least 60 folds. Nascimento, et al. [2000] successfully optimised the operating conditions of a nylon-6,6 polymerisation process. The pure ANN model for the process was developed using data generated from a rigorous semi-mechanistic model that had been fitted to the plant data. Optimisation was performed off-line by mapping all the possible solutions within the region of interest using the ANN model and locating the optimum using a grid-search method. Nevertheless, both works did not study different types of ANN structures to best model the processes; the ANN models used in both works are feedforward multi-layer perceptrons (MLP).

Hybrid FP and ANN models have also been widely applied in chemical engineering applications. Several methods of integration that were successful in certain range of applications had been proposed, but there were no specific proven rules. Published studies on this type of models included those carried out by Thibault et al. [2000], te Braake et al. [1998], Wilson and Zorzetto [1997], Thompson and Kramer [1994], Schubert et al. [1994], and Psichogios and Ungar [1992]. There even is less published work in chemical engineering on models that imbed prior knowledge into the system structure. A published work on this type of model has been done by Chang and Mavrovouniatis [1992].

1.3 OBJECTIVE

ANN models developed for RTO are different from those developed for process control or other off-line applications. RTO requires steady-state models that can yield all output variables required by the optimiser [Naysmith, 1997]. For large, multivariable processes, there can be more than 100 variables to compute. Since the application is on-line, the models must also have short computation times.

As mentioned previously, simple ANN models have successfully been used for RTO. However, there was no discussion on the choice or development of the multivariable ANN based models. There is a need to study the various types of ANN models to determine which are suitable for RTO, especially since there are difficulties to model chemical processes that may not be modelled using simple multi-layer perceptrons or can be more efficiently represented by other types of ANN. Different types of ANN models that had been studied in chemical engineering thus far were not developed for RTO applications; in fact, more complex models were mostly developed for control purposes. In addition, there has been very little work on imbedding prior information of a process in the structure of the network model.

The main objective of this research was to investigate, develop, and analyse different connectionist models that are appropriate for RTO applications. Because of time limitations, this research work concentrates specifically on exploring and developing suitable types of standard ANN and grey-box ANN models, without completing the whole RTO loop. In addition, there has been a previous work by Altissimi et al. [1998] that applied a standard ANN model in a complete RTO loop, where a reduction in computation time by at least sixty times was obtained. However, they did not explore the use of different types of grey box ANN structures, which is deemed to be necessary because there are chemical processes that cannot be modelled using standard ANN models. A grey box ANN structure is preferred over semi-mechanistic model because grey box ANN models would preserve the structure of ANN

models, which can be easily and efficiently solved. Even if a complex process had a series of grey box ANN models, the model would be solved in a straightforward manner, and would require much smaller computation time compared to models that were mechanistic in structure. For ease of investigation, the connectionist models were first developed for a simple but realistic flash process, followed by development for a complex crude oil distillation column.

This work is also meant to study the possibility of using different configurations of Hierarchically Structured Neural Network (HSNN) for modelling chemical processes. Prior information is imbedded into the HSNN structure to allow better modelling. So far, HSNN models have never been reported for modelling chemical processes. There have also been fewer studies on embedding simple and readily available prior information in the model network structure in chemical engineering applications.

The pure ANN models form the basis of comparison for the grey box ANN models that were developed. The grey box ANN models were imbedded with prior knowledge in the form of the architecture of the models. In addition, FP models were included in the hybrid ANN-FPM models. The models have mostly dominant ANN structure, which means that they have the desired characteristics that are suitable for implementation in RTO. The characteristics are:

1. Short execution time (inexpensive computation).
2. Good generalisation capabilities.
3. Robust (does not fail and has good convergence) and stable.
4. Easy to develop, update and maintain.

1.4 SIGNIFICANCE OF RESEARCH

This research presents the design of different types of ANN and grey-box ANN models with the desired characteristics for implementation in RTO. Coming up with a suitable model is important because this is usually the major stumbling block for on-line implementation. Although there has been some work done on implementing an ANN model in RTO, there has been no work done on developing and evaluating different ANN models in the literature. This work would therefore provide alternatives for other types of ANN and grey box ANN models, especially when simple ANN models are not suitable, or when prior information can be added to yield a better model.

Using grey box ANN models will provide options and possibilities of utilising existing information that can aid in increasing the accuracy of the model. Prior information that can be imbedded does not have to be in the form of equations as is currently performed in most research for modelling chemical processes. Information about suitable output variable combinations, or which of the output variables should be predicted first in a series of network models, can aid in increasing the accuracy of a model. This information is usually readily available, but not normally exploited in model development. In addition, incorporation of prior information can be achieved without involving complex procedures, and this is especially important for industrial implementation.

One of the grey box ANN models, the Hierarchically Structured Neural Networks (HSNN) based models, can be utilised when to imbed simple prior information into the structure. Completely driven HSNN can be used in the presence of an input variable that has a strong influence on a particular output variable of a system. Output-tuned HSNN can be used to imbed simple constraints that are usually known for chemical processes. This type of network, which has not been used to model chemical processes, has the potential to handle discontinuities in a system. This is an advantage over most ANN models because some chemical systems are known to exhibit discontinuities.

Developing ANN models for the simple, non-linear process and the industrial process served as an illustration of the different strategies in handling multiple input and output variables in different chemical systems. The ANN models developed here were for the complete processes, rather than for just specific variables, which is mostly the case in most published works.

1.6 THESIS OUTLINE

The following is the outline of the thesis.

Chapter 2 presents a literature review on the current research and general backgrounds on ANN and grey box ANN models that may be applied to chemical processes. Some of the different model structures have been applied to chemical processes, while others, with possible potential for future applications, have not. This chapter also provides a brief review on connectionist modelling.

Chapter 3 presents a description of the two processes (simple and industrial), modelled in this work. The simple but realistic system, a methanol-water flash drum, was used to study different ANN and grey box ANN models. The industrial process, a crude oil distillation column, is a practical candidate for RTO due to variations in feed and operating conditions, as well as having a complex physical model.

Chapter 4 presents the different ANN and grey box ANN models developed and tested in this work. There are two types of standard ANN models and three types of grey box ANN models investigated. Brief descriptions of the algorithms and structure of the models are also given in this chapter.

Chapter 5 presents the results and discussions of the different models described in Chapter 4. The models were tested on the flash systems and the crude oil distillation column. A comparison between the models and between the two different chemical processes was also made.

We conclude in Chapter 6 with comments on the current study and recommendations for possible future enhancements of the techniques developed here. Major contributions of this research are also given in this chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

Mathematical models are very important in chemical engineering, both in the design and operation of chemical plants. In plant operations, models are required for analysis, control (especially advanced control) and optimisation. Developing a reliable model for a chemical process is a major obstacle in implementing advanced control and optimisation because of the complexity and cost involved [Hussain, 1999]. Consequently, there is much research in developing different process modelling techniques.

Real time optimisation (RTO), which is the continuous evaluation and adjustment of a process operating conditions to optimise the economic productivity subject to constraints, traditionally requires rigorous steady-state plant models. These models are difficult and expensive to develop and maintain because of the skill and time requirements [Naysmith, 1997]. Currently, there are efforts to seek other types of suitable models, such as off-the-shelf commercial simulation packages or purely black box models like artificial neural networks (ANNs). A previous work found using commercial simulation packages for RTO to be impractical [Naysmith, 1997]. Several works suggested the use of ANN as the process model for RTO [Thompson, et al., 1996; Thompson and Kramer, 1994]. In one work, an important variable was estimated using an ANN model as part of a larger rigorous model for use in on-line supervisory

optimisation [Sabharwal, 1997]. In the works of Altissimi, et al. [1998] and Nascimento and Giudici [1998], rigorous FPMs were used to generate training and testing data to develop ANN models for a chemical process to be used for optimisation. In both studies, the ANN models were found to be accurate and were able to cut down computation time, which is very important for on-line applications.

The current work aims at exploring an alternative approach for the process model of RTO. Because of setbacks in using purely FP models, this work concentrated on ANN models, and grey-box ANN models that are able to incorporate prior knowledge into neural network models in the form of hybrid neural networks – FP models, or modular neural networks, or a combination of the two.

In this chapter, a review of RTO is presented to understand the problems and requirements in implementing RTO, especially on the role of the process model. A review on the previous work on RTO is also given. Then, an overview of ANN is presented to provide the basis of ANNs along with their strengths and their weaknesses. Different types of the networks that have the potential to be used in RTO are also described. Background information on ANN is given in Appendix A. Next, different hybrid ANN-FP models reported in chemical engineering applications are surveyed. Finally, modular neural network architectures with potential applications to RTO are presented.

2.2 REAL TIME OPTIMIZATION

2.2.1 Introduction

Real-time optimisation (RTO) or on-line optimisation is the periodic update of process operating conditions, such as flowrate, temperature and pressure setpoints, so that the process is operating at its economic optimum, while at the same time fulfilling the

process and production constraints [Jang, et al, 1987]. RTO can be applied to a single unit operation, or even to a whole plant. There is great interest in RTO in industry because of stiff competition and increasingly stringent product requirements [Cutler and Perry, 1983]. In addition, advancements in computing power have enabled cost-effective implementation of RTO.

RTO is most beneficial for processes with a wide range of operating conditions. In industrial operations, variations in the operating conditions are quite common due to varying feedstocks, product specifications and prices, and economic trade-off [Naysmith and Douglas, 1995]. For example, in an oil refinery, the atmospheric distillation tower receives varying qualities of crude oil. Temperature set points in the column, which determine the cut points for the side draws, varies with different assays. This leads to complications in determining the desirable operating conditions. As such, implementation of RTO can lead to significant improvement.

Figure 2.1 illustrates the general structure for RTO based on the simulated implementation studied by Naysmith [1997]. Referring to this general configuration, the steps in a complete RTO implementation cycle are:

1. **Steady state detection**: this is where the plant data are monitored for a pseudo-steady state condition before allowing for optimisation to take place. Once an approximate steady state is detected, the measured process variables are relayed to the optimiser.
2. **Data reconciliation**: this part consists of gross error detection and data reconciliation. Gross error detection filters out overly erroneous data, while the rest of the data are reconciled with mass and energy balances from the process model.
3. **Parameter estimation**: the reconciled data are used to update model parameters, which are then used in the optimiser process model.

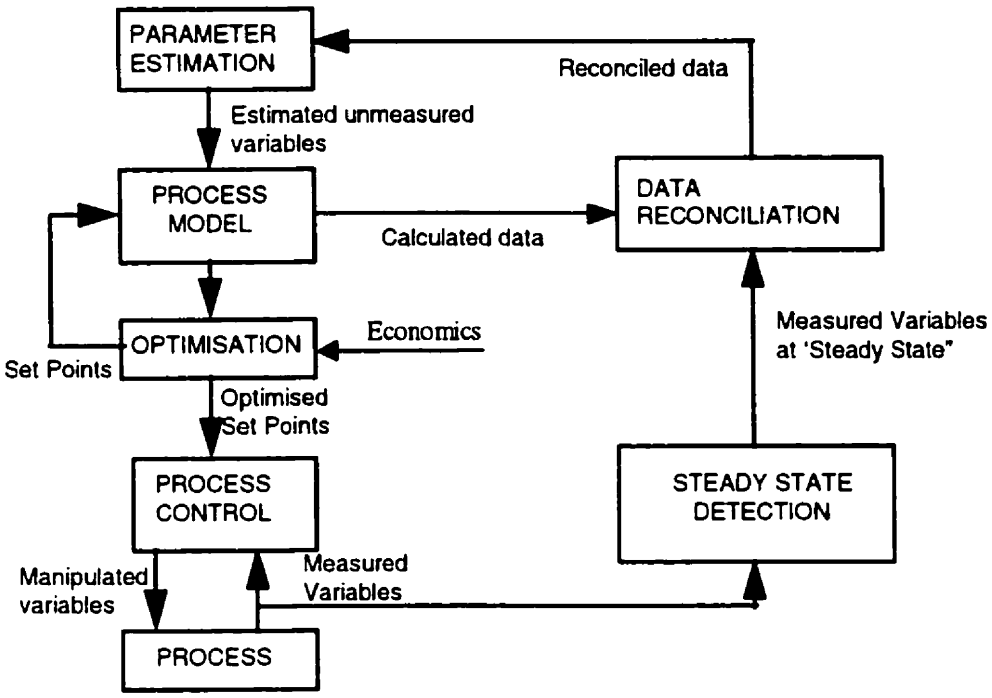

Figure 2.1: General configuration of a real time optimiser.

4. **Process model**: this is the most crucial component of the RTO. The process model calculates changes in the manipulated variables due to the updating of set points by the optimiser. The model is also needed in reconciling the measured plant data.
5. **Optimisation**: the optimum set points are searched subject to an economic objective function taking into account process constraints. The set points are iterated between this optimisation routine and the process model. Once optimised, the set points are passed on to the process control unit of the plant to be updated.

In general, industrial RTO is difficult to develop and maintain. The process models commonly used in RTO are rigorous first principles steady-state models, which

require extensive knowledge not only to develop, but also to numerically solve, and finally optimise. Consequently, developing RTO applications is extremely time-consuming and requires highly skilled and specialised staff. Maintenance of RTO applications is required when there are process modifications [Naysmith, 1997].

2.2.2 Previous Research

Naysmith [1997] successfully simulated the implementation of real-time optimisation (RTO) of a stabiliser column and a splitter column in commercial simulation packages. Using commercial packages to develop and hold the process model for RTO was investigated as a possible method to reduce the complexity of developing and maintaining RTO. Two simulation packages with different modelling and simulation approaches were compared: sequential modular (Aspen Plus) and equation oriented (Speedup).

Naysmith [1997] found several advantages and disadvantages in executing RTO with both the packages. These include but not restricted to:

- **Developing and debugging the model**. The Aspen Plus model is simpler to develop and debug compared to the Speedup model. Aspen Plus provides a graphical user interface and useful messages in debugging the simulation. Speedup is difficult to use and does not provide useful messages in debugging, especially in the event of convergence failure.
- **Execution time**. Speedup is computationally more efficient than Aspen Plus because all the model equations are solved simultaneously while Aspen Plus solved each module in a sequence. In the event of convergence failure, however, Aspen Plus is able to give more realistic results than Speedup because of the different algorithmic approaches in the simulation in the packages. Nevertheless, simulations

in both packages are computationally expensive, which limits the suitable choice of the optimisation method.

- **Optimisation**. Both packages use successive quadratic programming (SQP) as the optimisation method. Aspen Plus utilises the feasible path approach, while Speedup utilises the infeasible path approach. Consequently, optimisation with Aspen Plus is more conservative and takes more steps to reach the same optimum reached by Speedup. Nevertheless, in the event of convergence failure, the result given by Speedup may be completely meaningless because it does not occur in the feasible region of the process.

Naysmith [1997] concluded that process simulation packages currently available are not practical for use in RTO. This is mainly due to the following:

- Convergence failure (either because of the simulation model failing to converge or SQP failing to find an optimum) would occur and disrupt the RTO cycle. Consequently, the RTO cycle had to be restarted.
- Very high computation resource requirements, making the model unsuitable for repeated on-line evaluations.

2.3 ARTIFICIAL NEURAL NETWORKS

Due to the problems encountered in Naysmith's work, an alternative form of model is sought. A viable option is to use artificial neural network (ANN) models. This section provides a review of the background on ANN.

2.3.1 Introduction to Artificial Neural Networks

Artificial neural networks (ANNs) have been designed on the premises of mimicking the complexities of the brain functions in an effort to capture (or at least partially capture) the amazing learning capabilities of the brain. ANN is a sort of parallel computer/processor designed to imitate the way the brain accomplishes a certain task [Willis, et al, 1991]. The smallest processing element of ANN is a neuron (also called node) which performs simple calculations. Using the nodes collectively with massive connections among them results in a network that is able to process and store information for mapping the network inputs to its outputs. With this capability, there are widespread interests due to on-going and potential applications in solving complex problems particularly in the fields of pattern recognition (especially in speech and image processing), classification, control, forecasting, systems identification and optimisation.

ANNs are not a solution for all modelling problems. Therefore, it is necessary to understand the strengths and limitations of ANN to determine their applicability for a particular problem. Baughman and Liu [1995] lists the following strengths of neural networks:

1. **Distribution of information over a field of nodes**. This feature allows greater flexibility and robustness because a slight error or failure in certain sections of the network will not cause the whole system to collapse.
2. **Ability of ANN to learn**. ANN is able to adjust its parameters in order to adapt itself to changes in the surrounding systems, for example by using an error-correction training algorithm.
3. **Extensive knowledge indexing**. This means ANN is able to store a large amount of information and access it easily when needed. Knowledge is kept in the network through the connection between nodes and the weights of the connections.

4. **Suitable for noisy and inconsistent data**. This is possible because each neuron in the network encodes a minute feature of the input-output pattern, and thus minimising the effects of inaccurate data. The overall feature is mapped only when the nodes are assembled and co-ordinated together into a single network.
5. **Imitation of the human learning process**. The network can be trained iteratively, and by tuning the strengths of the parameters based on observed results. After repeated training and adjustments, the network can develop its own knowledge base and determine cause and effect relations.
6. **Potential for on-line use**. Once trained, ANN can yield results from a given input relatively quickly, which is a desired feature for on-line use.

Baughman and Liu [1995] also lists the following limitations of ANN:

1. **Long training times**. Training times for ANN can take too long, especially for large networks, to make the ANN impractical.
2. **Large amount of training data**. ANN needs large amount of input-output data for proper knowledge extraction. Therefore, if there are only a small amount of input-output data available, ANN may not be suitable for modelling the system.
3. **No guarantee to optimal results and reliability**. Although the network contains parameters that can be tuned by the training algorithm, there is no guarantee that the resulting model is perfect for the system. The tuned model may be accurate in one region and inaccurate in another. In addition, there is also the problem of getting trapped in local minima during training, resulting in less than optimal results.
4. **Difficulty in selecting good sets of input variables**. Selection of input variables is difficult because too many input variables will lead to large networks with too many parameters, which can in turn cause overfitting and poor generalisation. Too little or inappropriate input variables will lead to poor mapping of the system.

The origin of ANN can be first traced to the early 1940's in a paper by McCulloch and Pitts on the modelling of neurons [Venkatasubramanian and McAvoy, 1992].

Current research in ANN comes from diverse fields, such as the more traditional engineering fields (e.g. electrical engineering, computer engineering, etc.), mathematics and the sciences (physics, chemistry, and biology), to medicine, psychology, and business management.

In chemical engineering, while there have been numerous successful applications of neural networks, there are also those who claim neural networks to be nothing more than a class of nonlinear parameter estimation techniques. While the criticisms were sometimes well founded, there is a need to remember that drawbacks, extreme expectations and negative reactions are the norm in the exploration of an emerging field [Venkatasubramanian and McAvoy, 1992]. Hence, there is a need to find suitable roles that can best exploit the capabilities of neural networks in the chemical engineering field.

Currently, works in chemical engineering on ANN are mostly in process fault diagnosis, dynamic process modelling and process control. Compared to the large number of literature found on dynamic modelling, there are fewer papers on steady-state ANN process models. Nevertheless, there has been lately an increasing trend for diverse application of ANN to model steady-state processes. Among them are:

- Pollock and Eldridge [2000] and Whaley et al. [1999] fitted ANN models to experimental data for prediction of height equivalent of a theoretical plate (HETP) and pressure drop for columns with structured packing. Compared to a traditional semi-empirical method, the ANN models were found to give more accurate predictions of experimentally determined HETP values.
- Elkamel, et al. [1999] developed an ANN model for a hydrocracking unit in an oil refinery was using plant data. The model was used for prediction of product flow and quality.
- Mandlischer et al. [1999] fitted ANN to experimental data to predict the enthalpy of vaporisation. The model was found to be just as accurate as two physical models, and was slightly more accurate at critical temperatures.

- Sharma et al. [1998] fitted ANN to vapour-liquid equilibrium (VLE) data. They found that ANN was able to model the VLE phase envelope better than existing equations-of -state, especially for highly polar mixtures.
- Altissimi, et al. [1998] developed ANN models for a hydrocracker outlet gas separation unit, which consisted of four distillation columns in series, for use in RTO.
- Sabharwal [1997] estimated contaminant composition in a xylene distillation column in a refinery in Japan using ANN models trained by both plant and simulated data, and then used in off-line process optimisation.
- Cheung, et al. [1992] used steady-state ANN models as soft-sensors to provide inferential measurement of two variables of a refinery fractionator.
- Baratti, et al. [1995] used ANN as soft-sensors for predicting product compositions for a butane splitter and a gasoline stabiliser in a refinery in Italy.

2.3.2 Network Topologies

Neurons (also called nodes) can be connected in several different topologies, the most common being feedforward and recurrent networks. The nodes are arranged in layers. As such, the network may contain a single layer, or more than one layer, in which case it becomes a multilayer network. Appendix A contains detailed descriptions of a neuron.

Multilayer Feedforward Networks. There are three types of layers: input layer, hidden layers, and output layer (Figure 2.2). There can be more than one hidden layer. The hidden layer extracts higher order information from the data. Inputs to the neurons in a layer come from the neurons in the preceding layer. In a standard multilayer feedforward network, all connections are weighted.

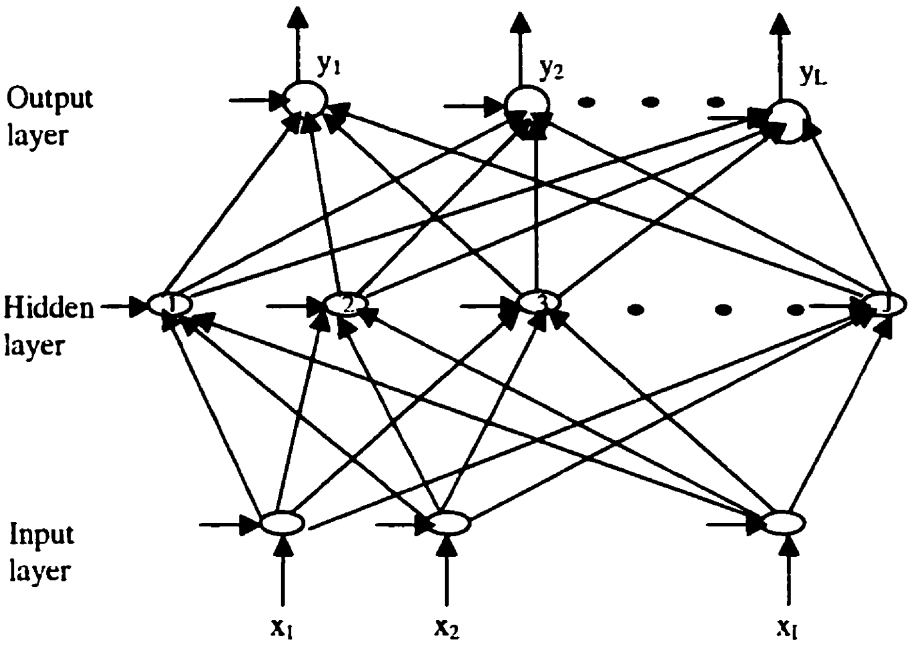

Figure 2.2: Schematic diagram of a three-layer feedforward network.

Recurrent Networks. A recurrent network is distinct from a feedforward network in that it has at least one feedback loop, as illustrated in Figure 2.3. Self-feedback, which is when the output of a neuron is fed back to its input, can also occur. Addition of unit delay systems or zero-order holds (denoted by z^{-1}) in recurrent networks, is very common in dynamic modelling.

2.3.3 Developing ANN Models

In general, developing ANN can be summarised into the following steps:

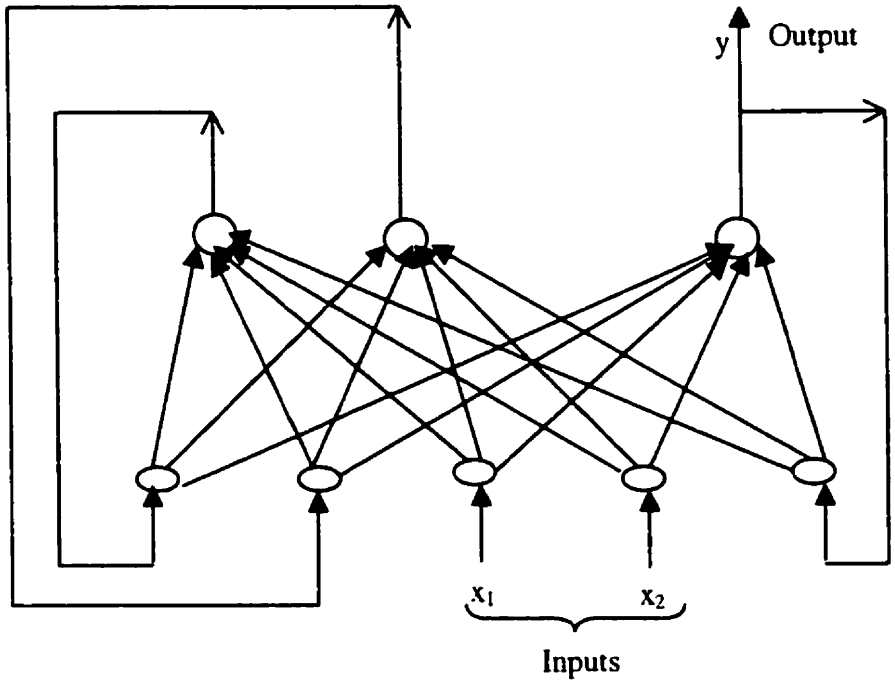

Figure 2.3: Schematic diagram of a recurrent network.

1. **Pre-processing**. Some pre-processing must be performed on the data. Although ANN is able to handle noisy data, industrial data that are very noisy should be filtered as much as possible. This is because data with too much noise can mask weak non-linearities [Cheung, et al, 1992]. Input/output data must also be scaled between 0 and 1 or −1 and 1 depending on the type of activation function used. This is very important to avoid saturating the activation function of the neurons. In addition, scaling the data will smooth out highly convoluted dimensions, making it easier for the network to learn the function surface [van der Walt and van Deventer, 1993].
2. **Training/learning phase**. Learning is the process of adjusting weight factors based on systematic and efficient trial and error. Training is the process of adjusting weight factors until the output patterns reflect the desired relationship [Baughman

and Liu, 1995]. To do this, the network is repeatedly presented with a set of known input/output data. The network learns the input/output response behaviour, and subsequently undergoes further training. This is the longest and most time-consuming step. It is also the most important to the success of the network. Common types learning algorithms are described in Appendix A.

3. **Recall phase**. The network is tested with the training data. Further adjustments on the weights are made, if needed.
4. **Generalisation**. The network is tested with data that it was not trained with before. This phase will determine the interpolation/generalisation capability of the network. Different numbers of nodes are tested because there is a possibility for the network to overfit and badly generalise.

Selection of suitable input variables is extremely important to properly map the relationship with the output variables. For large multivariable systems such as those encountered in chemical engineering, this is a difficult task. A major deterrent in using neural networks in chemical engineering is when the number of input variables is large, primarily due to what is called the "curse of dimensionality" [Wang, et al. 1995]. This is due to the fact that as the number of input variables increases, memory storage and computational cost increase exponentially. The number of parameters to be estimated will also increase, leading to poor generalisation capabilities [Sridhar, et al, 1998].

There have been a number of research activities on designing methods to select relevant inputs and thereby reducing the input dimension. In one of their works, Bhat and McAvoy [1992] built a network including all the input variables, and then started to prune them out one by one until all irrelevant inputs were eliminated. In another work, the opposite strategy was taken by growing the network size starting with an input pattern that is considered to be the most important [Sridhar, et al, 1998]. In these works, however, the network must be designed and trained before the relevant inputs can be chosen. A method put forward by Sridhar et al. [1998] enables the identification

of important variables before the ANN model is developed. The method, which is known as Information Theoretic Subset Selection (ITSS), is based on information theory. The ITSS method allows the estimation of the percentage of total information in a subset with respect to the entire input vector. Input vectors with large percentage of information can then be selected to develop the ANN model.

2.3.4 Multilayer Perceptrons

Multilayer perceptrons are feedforward multi-layered networks that are capable of performing just about any linear or nonlinear computation and can approximate any reasonable function arbitrarily well. Back propagation learning algorithm is one of the earliest and most common method for training multilayer perceptrons. It is used to train nonlinear, multi-layered networks to successfully solve difficult and diverse problems such as perform function approximation, pattern association and pattern classification, non-linear system modelling, time-series prediction and image compression and reconstruction [Hassoun, 1995]. Refer to Appendix A for a detailed description of backpropagation learning.

Leonard and Kramer [1990] showed that the backpropagation algorithm is inefficient and has poor convergence on serial processing machines (ie. computers). Backpropagation learning is generally slow because of the characteristics of the error surface that is characterised by numerous flat and steep regions and has many troughs that are flat in the direction of search. In addition, there are local minima at error levels above the levels of the global minima of the surfaces. This causes the back propagation learning to become stuck at the local minima and converge very slowly [Lin and Lee, 1995]. To speed up the performance of backpropagation many enhancements and modifications have been proposed [Lin and Lee, 1995]. Details of the recommended modifications are given in Appendix A.

Other than backpropagation algorithm, there are currently many other techniques that can be used to train MLP. Among them are the Levenberg-Marquardt and the conjugate gradient training algorithms.

2.3.5 Radial Basis Function Network (RBFN)

RBFN is based on the concept of the locally tuned and overlapping receptive fields that exist in the cerebral and the visual cortex [Moody and Darken, 1989]. The receptive fields of the network are radial basis functions, which can be adaptively tuned to provide sufficient overlapping for smooth mapping, but sharp enough for good approximations.

Well known for its fast, localised training, simplicity and generality, the network attracted much research, especially in the late eighties and in the nineties. The network performs very well for classification and multidimensional curve-fitting (approximation) problems. RBFN is also suitable for on-line applications because it can be rapidly trained [Freeman and Saad, 1997]. Among the applications are speech recognition, image processing, fault diagnosis, process control, time series analysis and general function approximation.

RBFN has a feedforward structure (Figure 2.4). It differs in terms of operation from the standard feedforward neural network in the following aspects:

1. The first layer connections to the second layer are not weighted.
2. The hidden layer has J nodes, usually with Gaussian density function:

$$h_j(x) = \exp[-\|x - c_j\|^2 / (2\sigma_j^2)] \qquad (2.1)$$

where c_j , $j = 1,\ldots,J$ are the RBF centers, and σ_j is the RBF width parameter. c_j determines the location and σ_j determines the span of the activation region of the nodes in the hidden layer. Each node in the hidden layer corresponds to a unique local neighbourhood in the input space. Within the region of activation, the closer

the input, x_i, is to the centre of the receptive field, c_j, the higher the activation level, with the maximum being one when x_i is at c_j.

Learning for RBFN is divided into two parts. The first part is on the synthesis of the hidden layer, while the second part is on getting the weights of the output layer. The separate training scheme exploits the localised presentation of the hidden layer units, since only the nodes activated by an input needs updating.

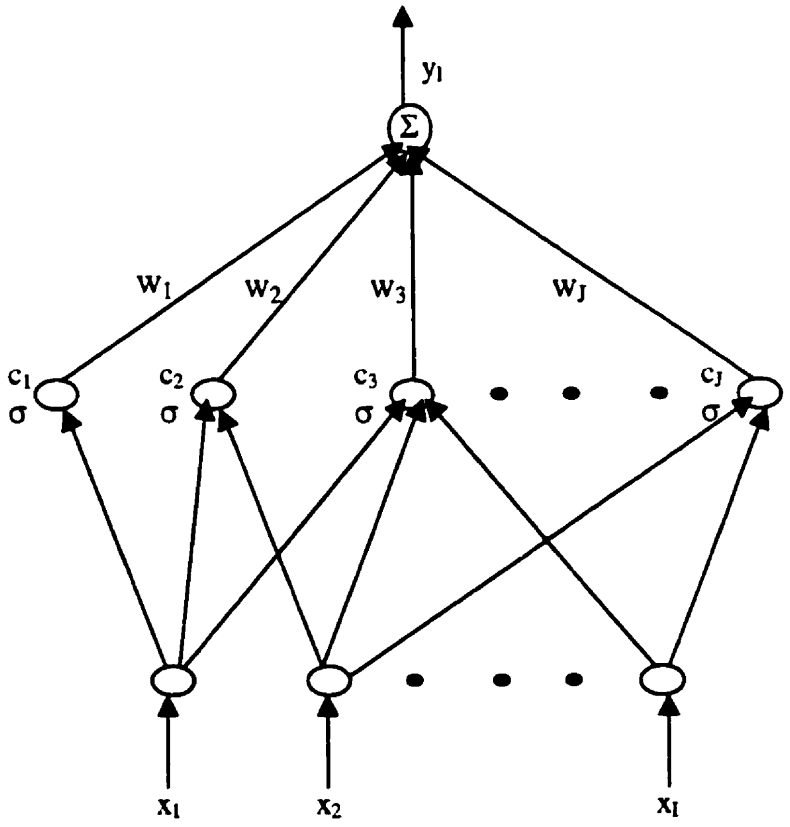

Figure 2.4: General structure of a radial basis function network.

The first part of RBFN training is to get the receptive fields parameters, which are the centers, c_j, and the width of the receptive fields, σ_j. Several learning schemes exist for determining c_j. The different approaches are discussed further in Appendix A. A single value of σ is sufficient to be used for all the receptive fields. RBFN with the same σ for each receptive field in the hidden layer was theoretically proven as universal approximator [Hassoun, 1995].

The second part of RBFN training pertains to the task of finding the weights of the output layer, and is fairly straightforward. The works surveyed used linear regression, singular value decomposition or one of the backpropagation algorithms, like the delta learning rule [Hassoun, 1995; Haykin, 1994, Leonard, et al., 1992].

Chen et al [1991] came up with the orthogonal least squares (OLS) algorithm, which has node-growing capability. The OLS algorithm provides a systematic method to select RBF centers. The centers are selected one at a time such that the approximation errors of the network are effectively reduced at each step. This recursive procedure is terminated once the errors have reached below a prescribed value. The MATLAB neural networks toolbox uses this algorithm to find the centers of RBF networks. An advantage of this method includes a smaller number of nodes in the hidden layer than that of RBF with randomly selected centers. Another advantage is the avoidance of numerical ill-conditioning frequently encountered in RBF with randomly selected centers.

Leonard et al [1992] introduced the validity index network (VI net), which is an extension of RBFN. In addition to the network output, the VI net indicates when the network is extrapolating. The network is able to indicate any extrapolation based on the estimation of the local training data density.

2.4 HYBRID ANN-FP MODELS

2.4.1 Background

Hybrid ANN-FP models were developed to overcome the disadvantages, while utilising the advantages of both approaches. They are designed with the aim to enable these two approaches to complement each other so that accurate and efficient models can be realised. In the literature surveyed, advantages of a hybrid model includes:

- Good generalisation and extrapolation capabilities [van Can et al., 1996].
- Easier and consume less time to develop than their rigorous FP models counterparts [Su et al., 1992].
- Able to extract physical interpretation from the model [Psichogios and Ungar, 1992].
- Accurate and reliable even when data is sparse and noisy [Thompson and Kramer, 1994].

te Braake et al. [1998] classified models with a mixture of FP (or white box) models and black box models (usually ANN) into two categories: semi-mechanistic models and grey box models. Semi-mechanistic models are based on FP models, with certain parameters or variables calculated from black box models or empirical data. The models are of the same form as the white box models. Grey box models are based on black box models, with prior knowledge being used to provide additional information. The models are of the same form as the black box model. Nevertheless, this definition serves only as a general guideline of notation for classification of these models, and is not necessarily used in other works that had been published. For example, Zorzetto et al. [2000] defined semi-mechanistic models to be the same as grey box models, which could also be called hybrid models. There are also models that can also fall into either category, like the model used by Nascimento et al. [1999]. On the whole, though, most

published works used the term hybrid model to generally indicate a model that is a mixture of FP models and black box (usually ANN) models.

Most applications of hybrid ANN-FP models found in the literature are designed for dynamic models, which are used for control and scheduling. Thompson and Kramer [1994] recommended steady-state hybrid models for RTO because of their potential advantages over traditional FPM and traditional ANN.

2.4.2 Structures of Semi-mechanistic Models

The most common structure of semi-mechanistic models found in the literature is the serial configuration, shown in Figure 2.5. In this configuration, the input variables, X, is fed to the ANN, which is used to estimate one or more parameters, O, that are difficult to obtain from a mathematical model. The outputs from the ANN, O, from the ANN are then used along with the inputs, X, in a FPM to calculate the process outputs, Y. The ANN models can be trained with either O or Y. An inherent assumption of this configuration is that other than the parameters estimated, the rest of the FP model is accurate. This is the most common hybrid structure seen in the literature. Examples in the literature include:

- microbial growth rate for a batch beer production [Zorzetto et al., 2000],

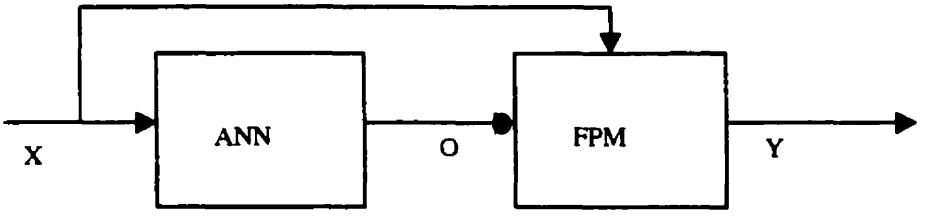

Figure 2.5: Serial ANN-FP hybrid model.

- specific reaction rates for a continuous stirred tank reactor [te Braake et al., 1999],
- concentration gradient, specific cake resistance and cake interface concentration for crossflow microfiltration [Piron et al. 1997],
- heat and mass transfer coefficients [Cubillos et al, 1996],
- microbial growth rate for a fedbatch bioreactor [Psichogios and Ungar 1992].

There are also variations in the application of the serial ANN-FPM structure. Schubert et al. [1994] added a fuzzy logic (FL) pre-processor to determine the different phases of a batch fermentation process when using ANN to predict the value of specific substrate consumption rate. Input data enters the FL pre-processor, which then directs the data to the proper ANN for the corresponding phase.

Fu et al. [1996] used a modification of the serial configuration. In all the previous serial semi-mechanistic models, supervised learning was used to train the network models. However, in the work by Fu et al., reinforcement learning was used. The performance evaluation unit (PEU), a knowledge-based tool, served as a critic that monitored the output of the hybrid model and gave evaluations based on an experimental database.

Additional variations found in the literature includes:

- Thibault et al. [2000] modified the traditional approach slightly by training the ANN model with the error of the output variables instead of the error of the parameters being modelled with ANN.
- Gupta et al. [1999] developed a two-level serial ANN for the prediction of several variables and parameters before finally feeding them to the FP model.
- Wilson and Zorzetto [1997] used a serial semi-mechanistic configuration as the model for a Kalman filter, which is a state estimator.

Other than the serial structure, there had also been work on a parallel configuration, shown in Figure 2.6. In this structure, the ANN is trained to predict the

residual between FPM and actual plant data [Su et al., 1992]. In this way, the combined hybrid model is able to accurately predict the process output. A major advantage of this configuration is that a less accurate and simple model is sufficient because the ANN can make up for the model discrepancies.

Thompson and Kramer [1994] proposed a combination of parallel and serial hybrid configuration, shown in Figure 2.7. The network in parallel to the default model estimates the residue in the absence of a good FPM. Beyond the range of the ANN training data, only the FPM is considered. In this configuration, the parametric output model enforces the process constraints upon the output variables.

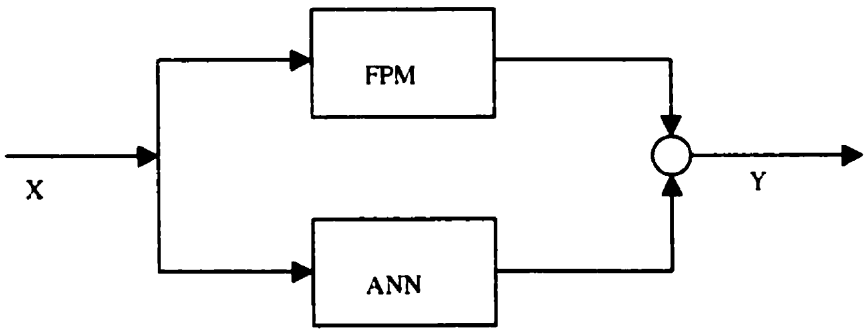

Figure 2.6: Parallel ANN-FPM hybrid model.

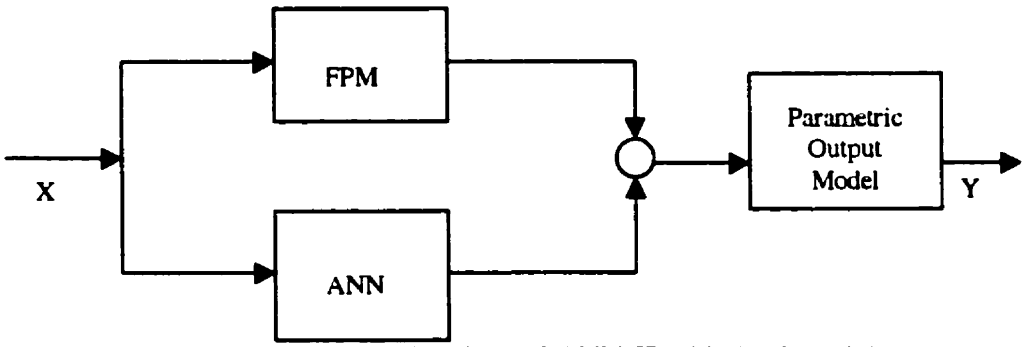

Figure 2.7: Combination of parallel and serial ANN-FPM hybrid model.

In most of the literature surveyed, the semi-mechanistic models were able to yield models that gave better results than the pure ANN models and other types of models. van Can et al. [1996] compared serial and parallel hybrid structure, as well as models with pure FPM and pure ANN for modelling and control of a pressure vessel. Using ANN models developed from experimental data, they found the serial configuration yielded the best results. They also showed that the serial configuration had better range and dimensional extrapolation capabilities than parallel configuration. On the other hand, Shene et al. [1999] found the ANN model more accurate than the serial semi-mechanistic model in modelling *Zymomonas mobilis* CP4 batch fermentation using experimental data.

2.4.3 Other Forms of Hybrid Models

Nascimento et al. [1999] developed a hybrid FP-ANN model for an industrial Nylon-6,6 polymerisation process to calculate the product relative viscosity, Y. The structure of the model is shown in Figure 2.8. Other than the input variables from industrial data, X, the hybrid model also had amine end-groups and carboxyl end-groups concentrations calculated from FP models, O, as inputs to the ANN model.

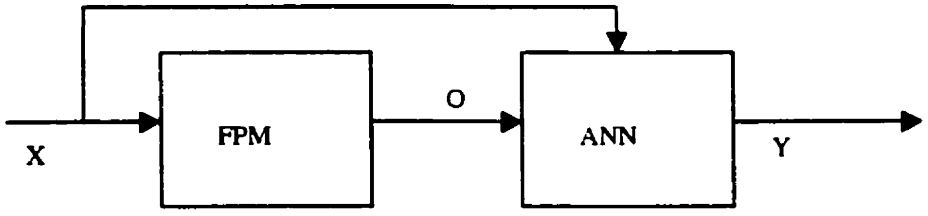

Figure 2.8: An alternative form of serial hybrid FP-ANN model.

The difference between this hybrid model and the serial semi-mechanistic model is that in this model, the final model is an ANN model, which receives additional information in the form of two additional input variables from the FP models. Furthermore, the FPM is used to calculate two output variables (the outlet concentrations), instead of a process or thermodynamic parameter. In a serial semi-mechanistic model, the final model is the FP model, which receives values of parameters predicted by ANN models.

2.4.4 FPM for Training ANN

There are several manners in which FPM has been used to train ANN to generate a black box model. In a sense, the FPM is integrated into the ANN model, but the resulting model is a purely black box model.

Thompson et al. [1996] suggested several ways to incorporate simulated data from FPM into ANN.

- Models of Models. The ANN is trained on data simulated using a simulation package. The main advantages are the fast and clean (i.e. noiseless) training, as well as fast on-line execution. Implementation on an actual plant was also reported.
- Combining Models. ANN models of different units in a plant that was trained on the simulated data are easier to combine than the actual physical models. The combined models are called a "metanet". This is useful for plant-wide RTO.

Sabharwal et al. [1997] used an accurate steady-state plant simulation to generate additional training data outside the range of available plant data for training an ANN model to be used for optimisation of a xylene distillation unit in a refinery in Japan. The range of applicability of the ANN model is therefore increased by the data generated through FPM simulations.

In another approach, Tsen et al. [1996] generated an augmented data set to train ANN. The augmented data are data interpolated and extrapolated using gradient information from FPM at actual experimental data. This procedure is advantageous when plant data for training are scarce.

Altissimi et al. [1998] generated data from a rigorous model simulated in Aspen Plus to develop steady state ANN models for a hydrocracker outlet gas separation unit, which consisted of four distillation columns in series. A reduction in computation time of at least 60 times was achieved when they replaced a rigorous FP model with the ANN model in the RTO loop of the gas separation process.

In a similar approach, Nascimento et al. [2000] generated data from a rigorous FP model to develop ANN models for an industrial nylon-6,6 polymerisation process in a twin-screw extruder reactor to be used for optimisation. The models were then used to map out the objective functions to execute a detailed grid search for finding the optimum in a specific region.

2.5 MODULAR NEURAL NETWORKS

2.5.1 Background

Theoretically, artificial neural networks (ANNs) are capable of learning complex and high-level systems, which are typical of chemical processes. Nevertheless, using a single, large ANN to solve large problems is prohibitive because of the computational complexities involved. One way to make the problem manageable is by using modular neural networks (MNN).

In general, MNN is based on the concept of divide and conquer [Jordan and Jacobs, 1994]. Large problems are divided into simpler, smaller and more manageable problems, solved (or conquered) using ANN, and are then combined to yield the solution. There exist different architectures of dividing, solving and combining MNN,

with various names like hierarchical networks, committee machines, stacked networks, and mixtures of local experts. In this work, MNN is used in a general context for systems of ANN that co-operate in an appropriate manner and the outputs combined to solve a complex system.

The design and use of MNN is motivated by the following setbacks in traditional large ANN [Chen et al, 1997]:

- A large network causes difficulties in training, such as difficulties in convergence and the problem of local minima.
- Slow training.
- No a priori orientation about the likely kinds of relationship between the input variables themselves and between input and output variables.
- Structure of ANN is unrelated to physical system even after training, providing no insight to the actual process.
- A large number of inputs causes a large number of parameters to be estimated, reducing the generalisation capability of the ANN.

The advantages of MNN compared to traditional ANN are [Gallinari, 1995]:

- Reduce model complexity, resulting in a more efficient model.
- Incorporate existing qualitative and/or quantitative knowledge into the network.
- Possible to decipher the relationship between the variables involved through connections among and within the network modules.
- Increase in robustness and flexibility.

2.5.2 Classes of Modular Neural Networks

Gallinari [1995] divided MNN into three classes, listed as follows:

1. Partitioning of the input space. The input data is partitioned into several subspaces. This class of MNN is the most commonly encountered in the literature. Each module of the network specialises in a section of the input space or in a task-specific function.
2. Successive processing. The whole complex problem is decomposed into specialised modules that are carried out successively.
3. Combining decision. Several different network models are used to process the input data and the outputs are combined to give a better overall model.

Several studies carried out in the area of MNN classes are briefly outlined in the following sections. However, in many instances, there has been an overlap between the classes of MNN. More detailed information on some of the algorithms is given in Appendix B.

Input Space Partition. The input variables are divided into groups, where each module would consist of small networks that are experts on a specific task or in a specific range of the inputs. Lu and Ito [1997] divided problem decomposition into three categories:

1. Explicit decomposition. In this type of decomposition, sufficient prior knowledge is needed about the domain and decomposition of the system. Jenkins and Yuhas [1993] used this concept and embedded prior information to efficiently control a truck to back-up a dock. Mavrovouniotis and Chang [1992] also decomposed the input space into subspaces that represent a specific small portion of a process, a particular phenomenon or a constraint, or a single time instant within a time interval. The subsets, therefore, are localised spatially, temporally, or conceptually. As in Figure 2.9, subsets are then combined to form subnets, which are further combined to form a hierarchy leading to the output of the network. The overall network is trained as a whole, based on its final outputs.

2. Class decomposition. The problem is decomposed into subproblems according to inherent relations among the training data. Developed for classification [Lu and Ito, 1997], this method needs only common knowledge about the training data.
3. Automatic decomposition. The decomposition takes place as the learning process progresses. This is the most computationally complex method, and as such, it is not very suitable for large-scale problems. However, this is the most general method since no previous knowledge of the system is needed. Most decomposition methods

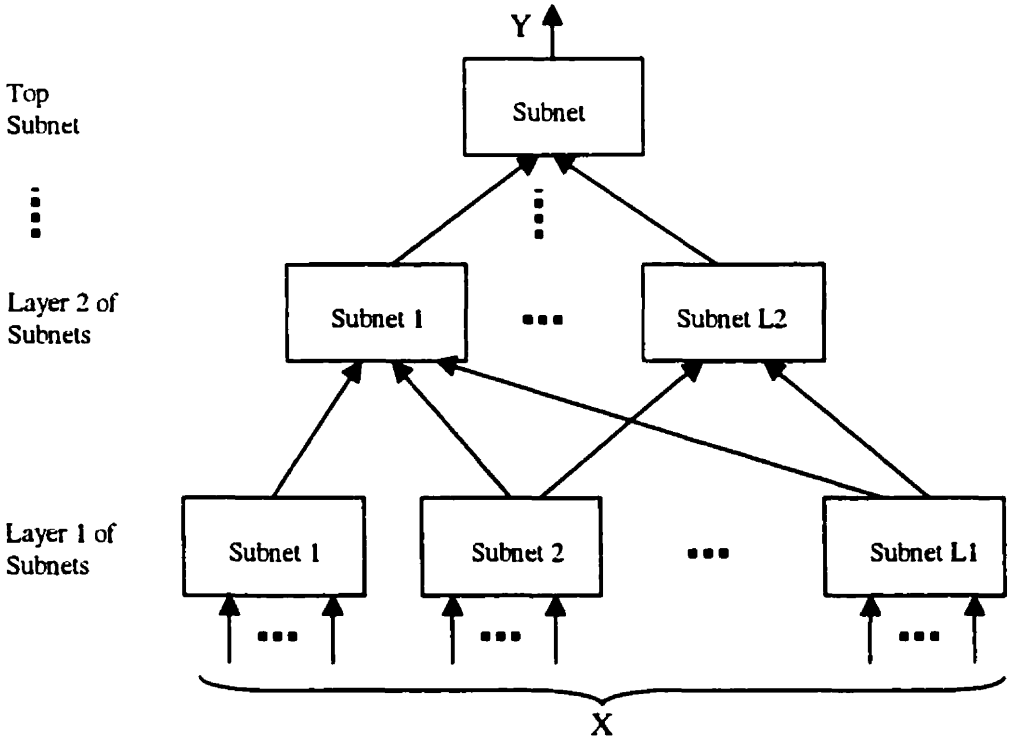

Figure 2.9: Structure of hierarchical network from Chang and Mavrovouniotis [1992].

fall into this category. For example, Chen et al. [1997] and Jacobs and Jordan [1994, 1995; Busson et al., 1998] used tree-structured architecture. Chen et al. [1997] used hyperplanes to partition the training data set into subsets. Soft partitioning was used so that there may be overlapping between adjacent partitioned input data subsets. The hyperplane is determined heuristically through a criterion, so that the resulting feedforward network can deal with two less complex subproblems with less computational cost. Jordan and Jacobs [1994, 1995] also utilised tree-like structure and soft partitioning for their mixture of expert networks, as shown in Figure 2.10. However, a gating network, which performed as the partitioning mechanism, was added to determine the degree of contribution from the outputs of the various local experts. The mixture of experts can also be arranged in a hierarchical manner. This algorithm had so far been applied for classification [Busson et al., 1998].

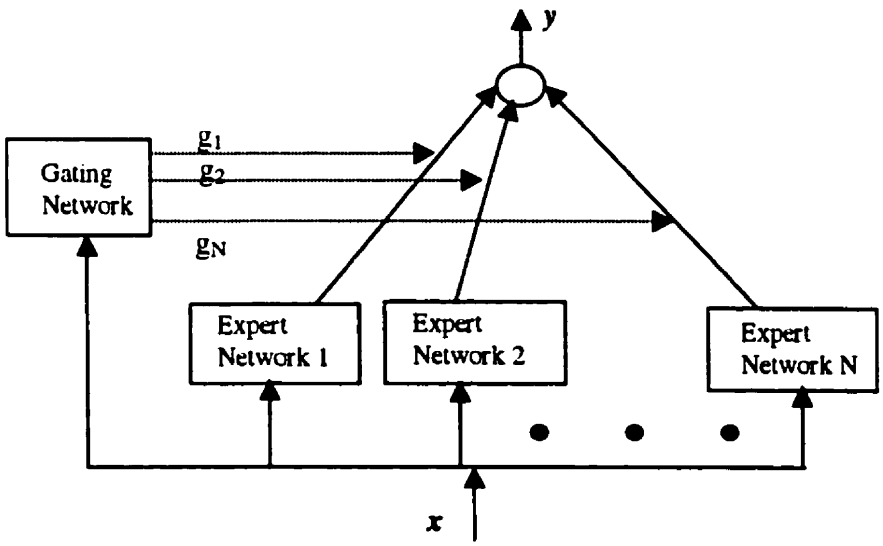

Figure 2.10: Mixture of Experts partitioned by a gating network as proposed by Jordan and Jacobs [1995].

Successive Processing. In this class of MNN, each successive layers of NN correspond to a specific processing of the data. This effectively breaks down the global system into successive tasks to be solved. In one work, Bittanti and Savaresi [1998] introduced two types of Hierarchically Structured Neural Networks (HSNN), which are divided into slave network and master network as illustrated in Figure 2.11. The first type, the Linear-Nonlinear HSNN, is characterised by the slave function being a linear combination of the slave inputs, with the network parameters provided by the master outputs, which are nonlinear. The output of the HSNN corresponds to the slave network output. This type of HSNN is completely driven, and is especially suitable for functions with discontinuity (i.e. where one of the variables has a switching effect), or for zeroing of functions due to certain variables. The second type, the Output-Tuned HSNN, is characterised by the master network providing only the gain and the offset at the output of the slave network.

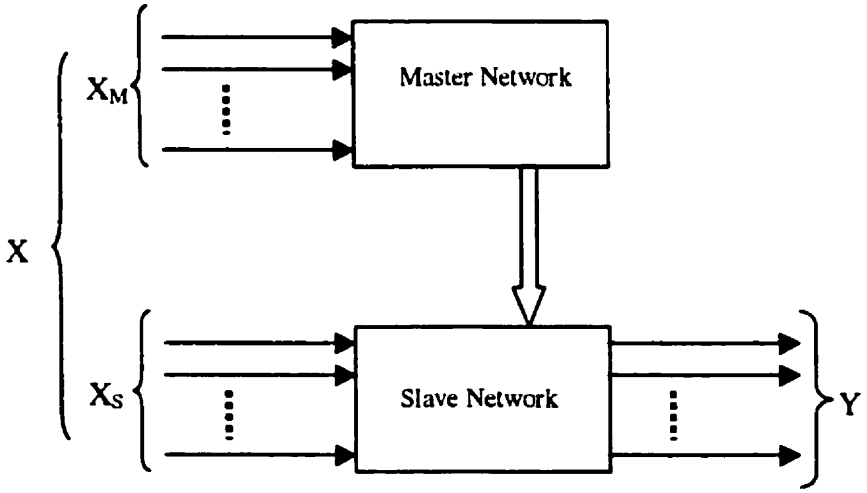

Figure 2.11: Hierarchically structured master and slave networks.

The network output is the sum of the slave output and one of the master outputs, multiplied by the other master output. This type of HSNN is partially driven and is especially suitable for fulfilling constraints in the form of a function. The hierarchical network proposed by Mavrovouniotis and Chang [1992] previously described may also be considered to belong to this class of MNN, because the subnets in each layer of the network represents a certain feature of the system, which at the end predicts the output of the MNN.

Decision Combination. The performance of networks can be improved by combining the outputs of independent modules. This may allow the network to have a larger range and lower prediction errors than a single large network. Examples of this type of MNN described in the literature are:

- **Stacked networks**. In stacked neural networks [Wolpert 1992; Sridhar et al., 1996], different independent networks, called level-0 models, are individually trained using part of the original data from the training set (Figure 2.12). The outputs of the level-0 models are then combined in a level-1 model to generate the network output. Sridhar et al. [1996] showed in their work that stacked networks were able to yield better results than traditional ANN models. Lanouette et al. [1999] found stacked networks to be useful for modelling processes when the number of data available for training and testing is small. Yang et al. [1999] successfully used stacked networks in the most difficult part of a multi-stage model of a semi-batch styrene polymerisation reactor. In another work, Zhang et al. [1999] uses stacked networks for predicting fouling in a batch reactor for the polymerisation of methyl methacrylate. The only setback is the large training time required for training for the individual network.

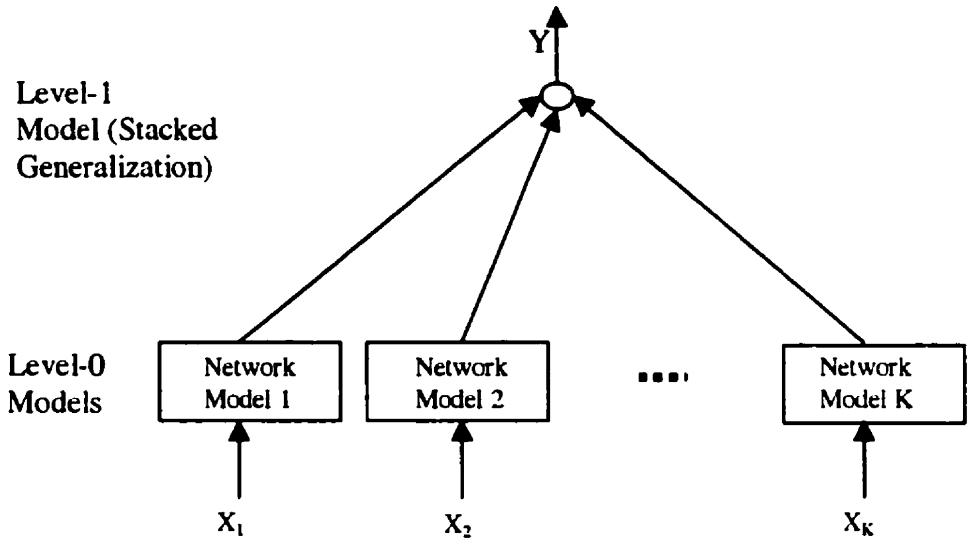

Figure 2.12: Structure for stacked networks.

- **Static committee machines**. Ensemble averaging networks is one of the static committee machines described by Haykin [1999]. The outputs of several independent networks trained with different initial conditions were combined, allowing an overall MNN with a larger range of initial conditions.

2.6 SUMMARY

For practical implementation of RTO, a process model must not only be accurate, but robust, easily maintained, and have short execution time. The use of pure FP models results in complexities in the development, maintenance and computation of RTO. Pure neural network models, on the other hand, are simple to develop and have fast execution of the model. However, using ANN may compromise accuracy, especially when

extrapolation is needed. ANN models also do not provide any of the process insights. In addition, large multivariable systems that are typical of chemical processes ultimately lead to neural network models that are too complex in terms of topology and training due to the "curse of dimensionality". Therefore, since models of chemical processes for RTO are multivariable, care must be taken when ANN based models are used for RTO to counter this problem.

Hybrid ANN-FP models combines both approaches of modelling to eliminate the weaknesses described and to take advantage of the strengths of both techniques. Several configurations exist in the literature, mostly in the form of simple parallel or serial structures. The configurations given are also mostly tested on dynamic models. Although Leonard et al. [1994] mentioned the possibility of using hybrid ANN-FP steady-state models for RTO, none has ever been reported in the literature.

Modular neural networks (MNN) models provide a possible solution to break up the multi-dimensional problem of chemical process models. In MNN, prior knowledge is incorporated into MNN models in the form of the structure, or by hard-wiring the network connection, although none of the works surveyed combined FP models with MNN. Nevertheless, algorithms for automatically decomposing the input space, which are mainly used for classifications, may not be necessary because of the extensive prior knowledge that is available on chemical processes.

The next chapter describes the two processes (simple and industrial) modelled in this research. The simple but realistic system, a methanol-water flash drum, is used to study different ANN and grey box ANN models. The industrial process, a crude oil distillation column, is a practical candidate for RTO due to variations in feed and operating conditions, as well as for having a complex physical model.

CHAPTER 3

PROCESS DESCRIPTION

3.1 INTRODUCTION

To develop and test different ANN and grey-box ANN models, two chemical processes were studied. The first process is a methanol-water (M-W) flash and the second process is a crude oil distillation column. From a practical point of view, the flash system can be solved accurately and quickly using existing FPMs. However, the M-W flash is as an ANN test case chosen because it is simple, yet realistic in representing non-linear, discontinuous, multi-variable chemical processes. This would enable a thorough evaluation and analysis of ANN models necessary to model this class of processes.

A flash drum splits the M-W feed into two outlet streams, a vapour and a liquid stream, with the vapour stream being richer in the more volatile component (methanol) than the liquid stream. A M-W system was chosen because the chemical non-ideality of the vapour-liquid equilibrium and heat of vaporisation of the mixture will add to the mathematical non-linearity of the system, which would be suitable to develop and test the models. At the same time, the M-W flash will be simple enough for studying different grey box models within a limited amount of time.

A crude oil distillation tower separates crude oil into many hydrocarbon products suitable for sale or further processing. A distillation column is actually a series of flash units stacked on top of one another. The crude oil distillation tower is an actual column

in the Petronas Penapisan Melaka refinery in Malaysia. The tower is suitable because it is a practical candidate for RTO, due to variations in operating conditions and has a very complex and large physical model. The feed to the column, crude oil, consists of hundreds of components and therefore, thousands of balance equations can be written for each tray of the column.

In this chapter, the methanol-water flash system and the Aspen Plus model developed for data generation are described. A detailed description of the crude distillation column and the Aspen Plus model are also presented.

3.2 SIMPLE PROCESS, THE M-W FLASH

A two-component flash column is a single stage separator, which splits the feed into a vapour, V, and liquid, L, product, as shown in Figure 3.1. Referring to the figure, z, y, and x are the compositions of the more volatile component, methanol, in the feed, vapour, and liquid streams, respectively. For a methanol-water flash, the vapour stream is richer in methanol because methanol is more volatile than water, and the liquid stream is richer in water for the same reason. A methanol-water mixture is non-ideal in the sense that the equilibrium constant, K, and the heat of vaporisation are functions not only of temperature, T, and pressure, P, as ideal solutions are, but are also a function of composition, x and y. This is a complicating factor.

In chemical processes, the complexity of a process model does not only depend on the physical equipment, but also on the chemical components involved. Although a process may use the same unit operation, the behaviour and thus the model of the process is different when different chemical components are involved because of the different thermodynamic behaviour of the mixture. Often, the unavailability of thermodynamic

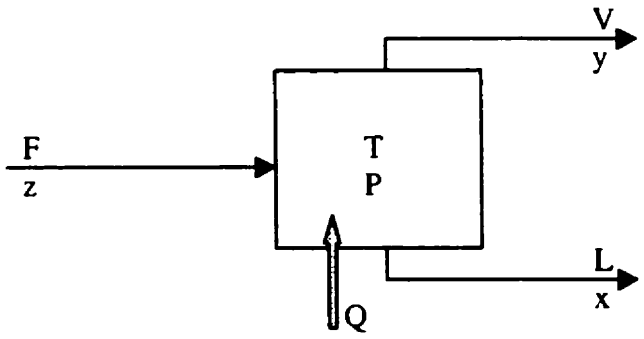

Figure 3.1: Schematic diagram of a general two-component flash column.

data of the components involved will make it difficult to develop good mathematical models even for a simple process.

With regards to vapour-liquid equilibrium for a two-component mixture, which is the simplest possible case, Figure 3.2 illustrates a so-called T-x-y phase diagram at constant P. T_A and T_B are the boiling point of components A and B respectively, where A is the more volatile component and therefore has the lower boiling point. The compositions plotted in the diagram are the composition of A. If a mixture with a composition of A, z_1, at T_1, then the mixture exists only in the liquid phase. A liquid mixture in this region below the solid bold line is known as sub-cooled. If the temperature of the same mixture is raised to T_{bub}, the first bubble of vapour will appear. Temperatures along this solid, bold line are the bubble point temperatures. A liquid mixture at the bubble point is called a saturated liquid. If the temperature of the mixture is raised to T_2, the mixture is now in the two-phase envelope. The mixture splits into a vapour phase with composition y_1 and a liquid phase with composition x_1 along the horizontal tie-line. As the temperature is raised to T_{dew}, the mixture will continue to

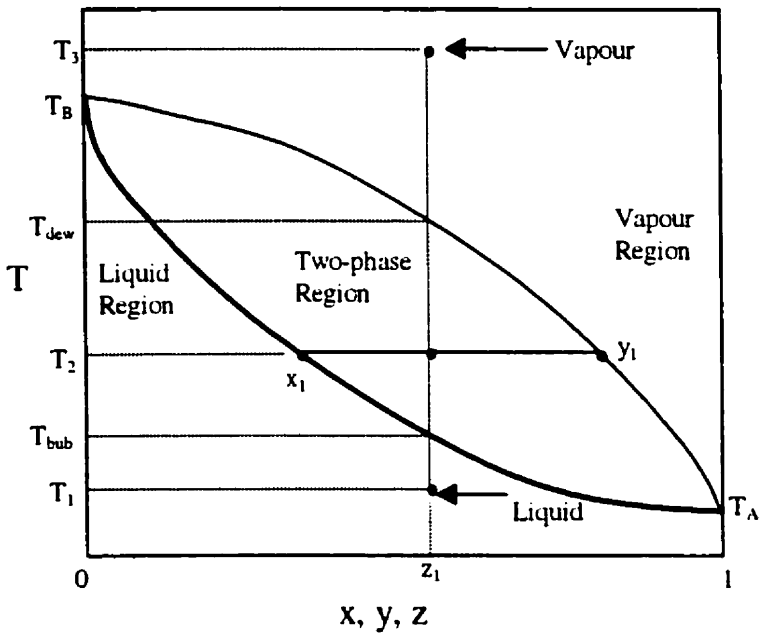

Figure 3.2: A general two-component phase diagram.

vaporise and at T_{dew} will exist only as a saturated vapour. Temperatures along this line are the dew point temperatures, at which the last drop of liquid exist. If the temperature of the mixture is raised to T_3, then the mixture exists only in the vapour phase. A vapour mixture in this region above the saturated vapour line is superheated. Consequently, as temperature is increased throughout the single-phase liquid region to the two-phase vapour- liquid equilibrium region and finally to the single-phase vapour region, there is a sharp discontinuity in the quantity and composition of vapour and liquid.

In this work, the process was simulated in Aspen Plus, a commercial steady-state process simulator. For data generation, feed stream temperature and pressure were kept constant at 1 bar and 60C, while the feed flow rate, F, feed composition, z, flash temperature, T, and flash pressure, P, were varied over the following ranges:

- F (110-250 kmol/hr)
- z (0.0909-0.6)
- T (70C-90C)
- P (0.7-1.1 bar).

The FLASH2 model in Aspen Plus can be used to model vapour-liquid flashes, evaporators, knock-out drums and many other two-phase single-stage separators. FLASH2 determines the thermal and phase conditions of a mixture of one or more inlet streams, when the outlet conditions are specified. The outlet conditions of the flash can be specified using two of four variables, which are, temperature, pressure, vapour fraction and heat duty. FLASH2 accepts any combination except vapour fraction and heat duty. In addition, it allows variation in the condition (e.g. flowrate, temperature etc.) of the feed stream to the column. Aspen Plus also has an extensive databank of physical and thermodynamic properties called Properties Plus. If the properties of certain components are not in Properties Plus, there are various estimation techniques in Aspen Plus which can be used in a process simulation.

To illustrate the non-linearity of the system, a general mathematical model of an isothermal flash is presented. The following equation relates the compositions to the flowrates of the system:

$$f(\psi) = \sum_{i=1}^{C} \frac{z_i(1-K_i)}{1+\psi(K_i-1)} = 0 \qquad (3.1)$$

where

z_i is the feed composition for component i,

$K_i = y_i/x_i$, is the equilibrium constant for component i,

$\psi = V/F$ is the fraction of the feed that goes to the vapour stream.

The equation can be iteratively solved using a root finding technique. However, for the methanol-water system, since $K_i = f(x,y,T,P)$, solving Equation 3.1 becomes more difficult because an initial guess or estimate of x and y will have to be made and matched

at each iteration. This problem can be solved by simultaneous iteration on (x,y) and ψ, or by having separate nested loops on (x,y) and ψ [Seader and Henley, 1998].

3.3 INDUSTRIAL PROCESS

The crude distillation tower chosen as the industrial process in this study was designed to process sweet crude oil (ie. crude oil with low sulfur content) and condensate. This process faces problems brought about by the varying composition of the crude oil and condensate feed. Crude oil composition varies from shipment to shipment, even if they come from the same well. Once in the storage tanks, the composition is not homogeneous because the oil is not mixed. Oil at the bottom of the tank would contain heavier components (ie. high boiling point and molecular weight) compared to oil at the top of the tank. Accurate feed composition measurement is almost impossible, even with off-line measurements. No on-line composition sensors exist for crude oil. For the chosen crude tower, the off-line measurements are inaccurate because the oil sample contains a high percentage of light components. Therefore, during sampling and analysis, the light components vaporise. Consequently, operators always have to adjust the set points for the operating condition of the process to meet the product specifications and yield. For instance, there are two different types of feed going to the crude tower, the condensate stream and the crude oil stream, which can be adjusted to keep the average feed composition at the approximate desired level. In addition, product quality is only measured in the laboratory once every twelve hours. This requires the operators to rely on past experience by observing the temperature and pressure profiles from the column to estimate whether the products are within specifications. Therefore, a good grey box model of the process, even one that is not used in RTO, will be very useful. This is because such models would be able to estimate the cold properties from the operating conditions, such as temperature profile of the column, and the average feed composition.

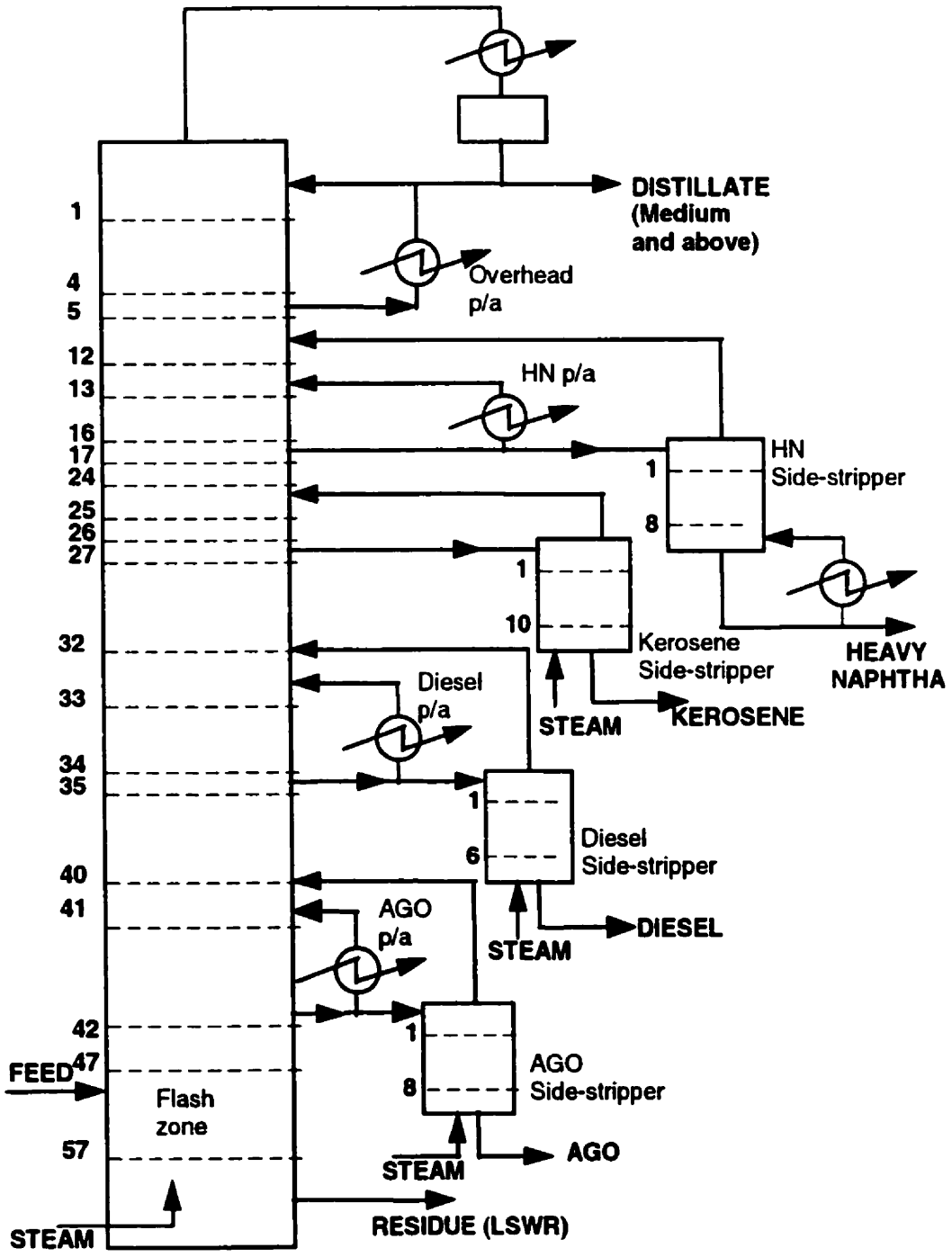

Figure 3.3: Crude tower flow diagram

Figure 3.3 illustrates a simplified flow diagram of the crude tower. There are four side strippers: heavy naphtha (HN) stripper, kerosene stripper, diesel stripper and atmospheric gas oil (AGO) stripper. Except for the HN stripper, which has a reboiler, all other strippers including the main column has medium pressure steam injected into the bottom of the column. Steam injected into the column reduces the partial pressure of the hydrocarbon, lowering the vaporisation temperature. The side strippers return the light ends that are withdrawn with the products in the side streams back to the main column. As such, the stripping rate in the side strippers can be used to manipulate the initial boiling point (IBP) of the product.

Feed entering the crude tower, which is a mixture of Bintulu condensate and Tapis crude, flashes into the flash zone, carrying over a small amount of LSWR, which is called overflash. This overflash allows some fractionation to occur on the trays just above the flash zone by providing internal reflux in excess of the sidestream withdrawal. At the same time, a slight amount of LSWR that enters the AGO side draw (also called run down) is desirable because of the resulting increase in AGO yield, as long as the quality is within specifications. Overflash is controlled by manipulating the coil outlet temperature (COT), which is the temperature of the oil leaving the furnace used for preheating, and AGO withdrawal. At constant COT, increasing AGO withdrawal will decrease overflash, while decreasing AGO withdrawal will increase overflash. An increase in COT will increase overflash and vice versa. There is a minimum overflash requirement of 3%. Other than AGO quality constraint, a high overflash will increase energy consumption in the form of heating fuel and pumping rate of the column pumparounds. Nevertheless, a low overflash will result in a dry AGO tray.

There are four pumparounds (p/a) in the tower: overhead p/a, HN p/a, diesel p/a and AGO p/a. In addition to providing liquid reflux and reducing the vapour loading on the upper part of the column, pumparounds also provide cooling to the tower. Thus, pumparounds can be used to control column temperatures, which affect product purity. Under normal operation, though, pumparounds are not used to control product quality or

column temperature, except for the occasional fine-tuning. An exception is the top column temperature, which is controlled by manipulating the top pumparound and reflux flow. Heat in the pumparound and product streams is recovered by heating the feed stream in the preheat train.

Product from the side draws must meet certain specifications. Operators obtain these specifications from the production planning section and adjust the tower operating conditions (ie. set points) to ensure on-spec products. The quality specifications are checked off line once during each shift - twice a day - at 06:00 and 18:00, and are thus called "cold" properties. Table 3.1 lists the specifications and the corresponding products and manipulated variables. It is important to take note of the cold properties because these are the values that would be predicted in the model output.

A reconciled steady state simulation of the crude tower has been developed in Aspen Plus using PETROFRAC, a rigorous tray by tray equilibrium based distillation column model designed specifically for petroleum applications. The main column, side strippers, pumparounds and condenser were all modelled as part of the column with PETROFRAC.

To obtain an accurate feed composition for the simulation, the products of the crude tower section were back-mixed and analysed. The feed assay information given to Aspen Plus were the true boiling point (TBP) curve, light ends analysis, stream specific gravity and average molecular weight. In Aspen Plus, the feed stream compositions were approximated with seven conventional components ranging from C2 to C5, and at least 50 pseudo-components. The Peng-Robinson equation of state, which is recommended for refinery applications [Aspen Technologies, 1995], was used to calculate all thermodynamic properties.

Table 3.1: Product specifications and manipulated variables of the crude tower.

	Specifications/ Properties	Manipulated Variables
Heavy Naphtha	IBP FBP	Top temperature or Q HN draw
Kerosene	Flash Point / IBP Freeze Point / FBP	HN draw SS Kerosene draw
Diesel	Pour Point / Colour IBP FBP	Diesel draw Kerosene draw Diesel draw
AGO	Pour Point / Colour IBP FBP	AGO draw Diesel draw AGO draw
LSWR	Pour Point	AGO draw

Note: IBP is initial boiling point
FBP is final boiling point
Q is reboiler duty
SS is stripping steam rate

3.4 DATA GENERATION

For both the flash system and the crude distillation tower, data were generated using the sensitivity analysis feature in Aspen Plus. In this feature, the independent variables specified were varied one at a time within the lower and upper limits. For example, if temperature, T, and pressure, P, were the independent variables, T was held at a constant value while P was varied. Once P reached the specified upper bound, T was incremented

to the next value and held constant while P was again varied. This procedure was repeated until the upper bound for T was reached.

Although the sensitivity analysis feature in Aspen Plus is convenient in generating a large number of data, the spread of data obtained is very poor for training neural networks, and the data is clustered around certain temperatures or other independent variables. To overcome the problem, a large number of data (at least three times the estimated amount that will be used for training) was generated. From this set, data collected at a certain fixed interval from the original data file were taken to be in the training set. This enables the training data to be sequentially ordered and have a better spread over the desired range. The orthogonal least squares (OLS) algorithm used to train the RBF networks performs better with sequentially ordered training data. Training data for all the models developed in this work are in sequential order, except for MLP models, which performs better with randomly ordered data. The same technique was repeated for generating testing data, but at independent variable values that were different from the training set. For example, for the M-W flash system, a total of 792 data points were generated, out of which 150 data points were selected for training. Another set of data was generated at different independent variable values, out of which 70 were selected for testing. The selection of the number of training data points was based on several trial runs meant to find the number that gave the best result in terms of the prediction errors and training times. A small number of the test data gave a poor estimation, while too many data points a large number led to a model overfit. As stated by Haykin [1994] on page 178 of his book:

> "When, however, a neural network learns too many specific input-output relations (i.e., it is overtrained), the network may memorise the training data and therefore be less able to generalise between similar input-output patterns"

Similar to the M-W flash system, the sensitivity analysis feature in Aspen Plus was also used to generate data for training and testing for the crude tower. Input variables for the ANN or grey box ANN models include the feed flow rates for the two feed streams, and the specified variables of a particular section for the tower operation. The output variables are the dependant variables that were needed by the optimiser and were calculated due to changes in the input variables. Ranges for the variables are within the operating region of the column. Within this region, the variables in each section of the column have negligible influence on other sections in the column, except the sections that are immediately above and below it. This allowed data to be generated one section at a time. Since the sensitivity analysis feature of Aspen Plus can only allow a maximum of five independent variables, any section with more than five independent variables were simulated one at a time for the different values of the sixth independent variable.

Table 3.2 lists the input and output variables of the network models for each section of the crude distillation column. Only variables associated with the particular section are included in the network model. Note that for the LSWR section of the column, the LSWR draw off was originally indicated as an output variable. This stream is considered as a waste stream; so, there was no strict specification or control. However, after generating data for the section, there was no significant change in the LSWR draw off rate (less than 0.001%). Therefore, the draw off rate was not modelled since it was essentially a constant.

The following lists the nomenclature of the input variables used in Table 3.2:

- Bintolt is the Bintulu condensate feed from the storage tank.
- Htfeed is the Tapis crude feed from the storage tank.
- HNdraw, Kerodraw, Diesdraw and AGOdraw are heavy naphta (HN), kerosene, diesel and AGO product draw off respectively.
- Qreb is the reboiler duty of the HN side stripper.
- SSK, SSD, and SSA are the stripping steam rates for the kerosene, diesel, and AGO side strippers respectively, and SSM is the main column stripping steam rate.

Table 3.2: Input and output variables for each section of the crude distillation column.

Crude tower section	Input variables	Output variables
Top of main column	Bintolt, Htfeed, HNdraw, Kerodraw, Qreb	Ttop, Ovhd, RR, Qcond, PAT
HN stripper	Bintolt, Htfeed, HNdraw, Kerodraw, Qreb	TtopH, TbotH, PAH, IBPH, FBPH, RhoH
Kerosene stripper	Bintolt, Htfeed, HNdraw, Kerodraw, Diesdraw, SSK	TtopK, TbotK, FPKero, IBPK, FBPK
Diesel stripper	Bintolt, Htfeed, Kerodraw, Diesdraw, AGOdraw, SSD	TtopD, TbotD, IBPD, FBPD, PourD, PAD
AGO stripper	Bintolt, Htfeed, Diesdraw, AGOdraw, SSA	TtopA, TbotA, IBPA, FBPA, PourA, PAA
LSWR (Bottom of main column)	Bintolt, Htfeed, AGOdraw, SSM	TBot, PourL

The following lists the nomenclature of the output variables used in Table 3.2:

- TtopH, TtopK, TtopD and TtopA are the top temperatures of the HN, kerosene, diesel and AGO strippers and Ttop is the top temperature of the main column.
- TbotH, TbotK, TbotD and TbotA are the bottom temperatures of the HN, kerosene, diesel and AGO strippers and Tbot is the bottom temperature of the main column.
- PAT, PAH, PAD, and PAA are the p/a at the top of the main column, and the HN, diesel and AGO strippers respectively.
- Ovhd is the overhead draw off rate.
- RR is the reflux ratio.
- Qcond is the condenser duty of the main column.
- IBPH, IBPK, IBPD and IBPA are the initial boiling point of HN, kerosene, diesel and AGO produced respectively.

- FBPH, FBPK, FBPD, FBPA are the final boiling point of HN, kerosene, diesel and AGO produced respectively.
- RhoH is the density of HN.
- FPKero is the flash point of kerosene.
- PourD, PourA and PourL are the pour points of diesel, AGO and LSWR produced respectively.

All the training and testing data for the M-W flash are given in Appendix C. Training and testing data for crude distillation tower are included in the enclosed diskette.

3.5 SUMMARY

The two processes studied to develop and test different types of ANN and grey-box ANN models are described in detail in this chapter. Method of data generation and the nomenclature used in both processes are also explained.

In Chapter 4, the ANN and grey-box ANN models developed and tested are presented. There are two types of standard ANN models and three types of grey box ANN models investigated. Brief descriptions of the algorithms and structure of the models are also given in Chapter 4.

CHAPTER 4

DEVELOPMENT OF ANN AND GREY BOX ANN MODELS

4.1 INTRODUCTION

In this chapter, the ANN and grey box ANN models developed and tested in this research are described. Brief descriptions of the algorithms and structure of the models are also included. All model simulations were performed in MATLAB using the neural network toolbox.

The models developed can be classified into two classes:

- standard ANN models
- grey box ANN models.

Two types of standard ANN models were compared to see which one is better as the base case:

- multi-layer perceptron (MLP),
- radial basis functions (RBF) network.

Three types of grey box ANN models were developed:

- hierarchically structured neural networks (HSNN),

- serial networks,
- hybrid network-FPM.

4.2 MULTI-LAYER PERCEPTRONS (MLP)

As mentioned in the literature review section, MLP are multi-layer feedforward networks. The MLP networks used here have an input layer, one or two hidden layers and an output layer. For all the models used here, the sigmoid function is chosen as the activation function of the networks because the training and testing data are normalised between zero and one. Two different types of feedforward training algorithms were used from the Neural Networks Toolbox of MATLAB: backpropagation with adaptive learning rate and the Levenberg-Marquardt algorithm. For both training algorithms, the number of layers in the network, the number of nodes in each layer, and the maximum sum of squared error that can be tolerated during training must be specified.

Traditional backpropagation algorithm uses the standard steepest descent algorithm with a fixed learning rate, μ, to find the optimal weights. The performance of the algorithm is highly dependent upon the value of μ. A value of μ that is too small will converge very slowly, while a value of μ that is too large will cause oscillations and instability. There is no single optimal value for μ because this depends upon the current position on the error surface that is being searched. Therefore, allowing μ to vary according to the error would improve the performance of the backpropagation algorithm. The adaptive learning rate algorithm in Matlab increases μ by 5% if the present error is less than the previous error. On the other hand, if the present error is more than the previous error by 4%, the present network parameters are discarded and μ decreased by 30%.

The Levenberg-Marquardt algorithm is well known for fast training. It has close to second-order convergence, without computing the Hessian matrix. Instead, the

Hessian matrix is estimated from the Jacobian matrix, which is much easier to compute than a Hessian matrix. Once a minimum is approached, the Levenberg-Marquardt algorithm in Matlab has a scalar parameter that is adjusted to smaller values, to increase the convergence rate of the algorithm.

4.3 RADIAL BASIS FUNCTION NETWORKS (RBFN)

The RBFN training algorithm in Matlab uses the orthogonal least squares (OLS) algorithm to iteratively select the centers of the radial basis receptive fields that will lower the network error the most. Unlike some RBFN training algorithms the number of centers (and thus hidden nodes) obtained using this training algorithm is less than the number of input vectors because the addition of the centers is stopped once the training error is less than a specified sum of squared error.

Other than specifying an error goal, the spread constant, σ, which determines the width of the receptive fields must also be specified. σ should be large enough for the receptive fields to overlap one another to amply cover the whole input range. Nevertheless, it should not be too large that there is no distinction between the output of different nodes in the same area of the input space. More detailed description of RBFN is in Appendix A.

4.4 HIERARCHICALLY STRUCTURED NEURAL NETWORKS (HSNN)

HSNN [Bittanti and Saveresi, 1998], uses prior knowledge of the output variable behaviour to divide the input variables into "master" and "slave" units. Both types of HSNN proposed by Bittanti and Savaresi were tested: completely driven and output

tuned HSNN. Two types of completely driven HSNN were developed: linear-nonlinear HSNN and nonlinear-nonlinear HSNN. A priori knowledge on the general relationships of the input variables with respect to the output variables, y, is needed to determine the master (U_M) and slave (U_S) inputs for all types of HSNN.

4.4.1 Linear-nonlinear HSNN

The master network for the linear-nonlinear HSNN is nonlinear, while the slave network is linear, as shown in the general structure illustrated in Figure 4.1. In this case, the variable chosen as the slave input should be approximately linear with respect to the output variable. Referring to the figure, V1M and V2M are the weights of the first and second layer of the master network respectively, and C1M and C2M are the biases. V1S and C1S, which are also outputs of the master network, are the weights and bias of the slave network. Also note that in Figure 4.1, ovals represent a standard neuron, while the output of a square neuron is the summation of the input multiplied by the weights. The network parameters are updated using equations that are derived based on backpropagation update formulas. Unfortunately, the update formulas given by Bittanti and Savaresi [1998] were ambiguous in some of the notations and had several errors.

The following update equations are based on a model with two output nodes in the master network as shown in Figure 4.2, which is the structure of linear-nonlinear HSNN that were mostly used in this work. The equations can be easily extended to have more or less nodes in the output layer of the master network.

The output of the HSNN (which is also the output of the slave network) is calculated as in Equation 4.1.

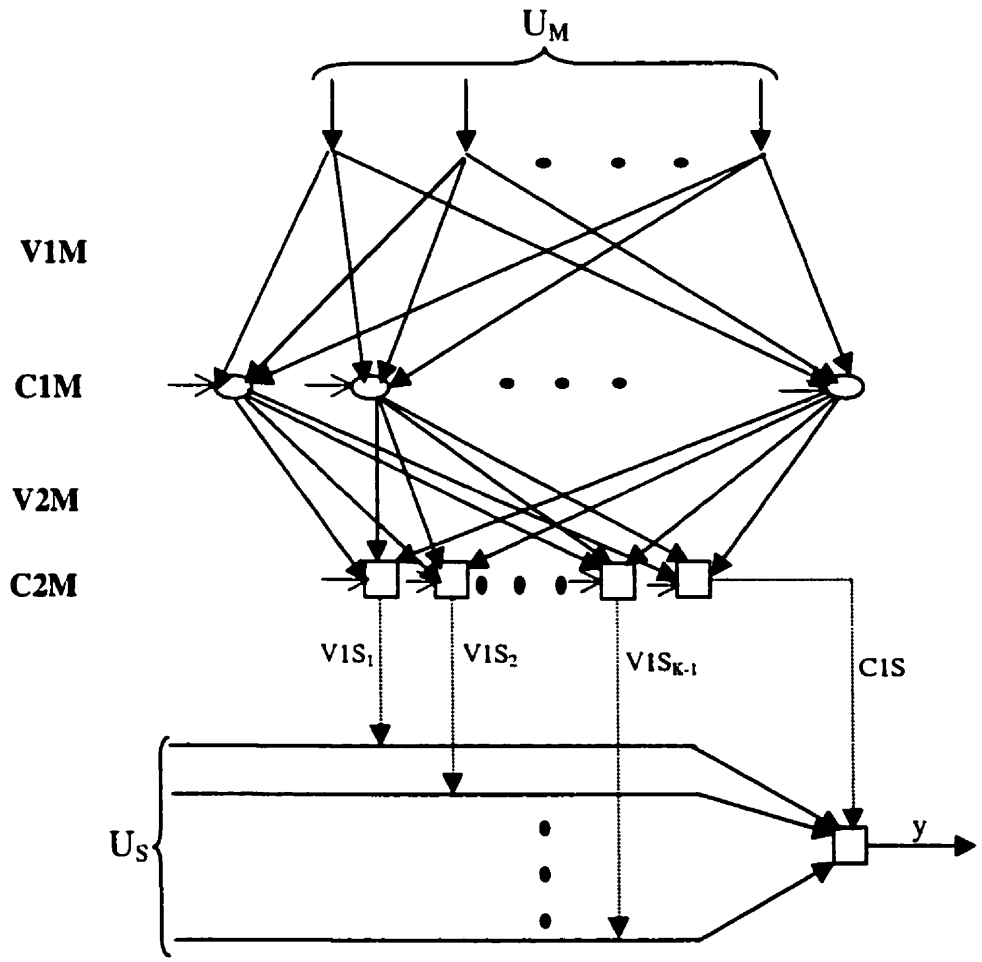

Figure 4.1: General structure of linear-nonlinear HSNN.

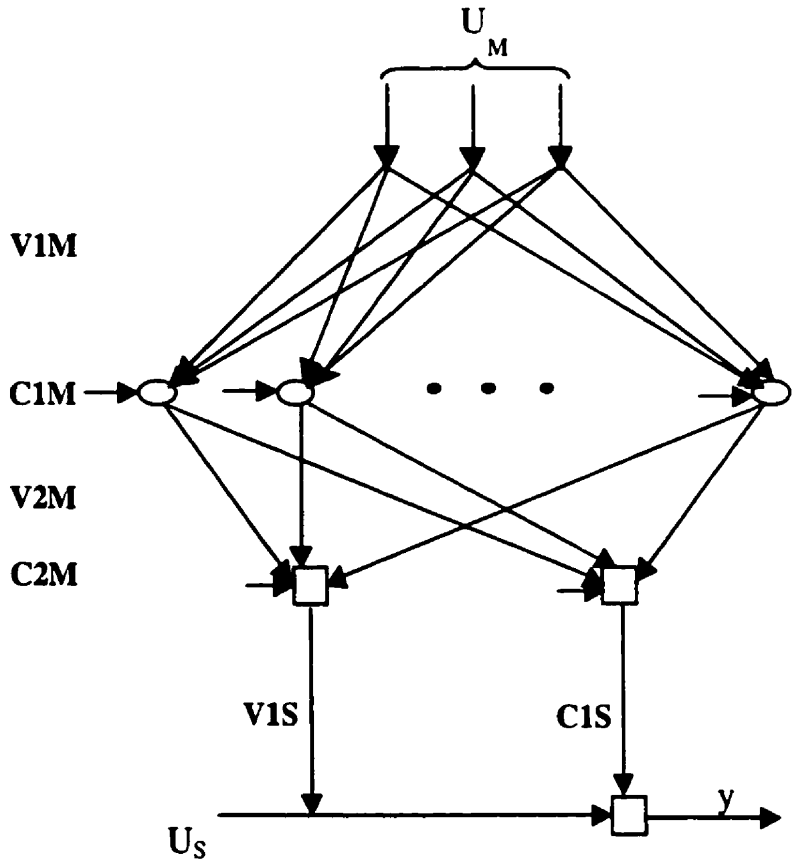

Figure 4.2: Structure of linear-nonlinear HSNN for M-W system.

$$y = \Sigma [(V1S)(U_S)] + C1S \quad (4.1)$$

where V1S = slave network weights and output of master network

C1S = slave network bias and output of final node of master network

Us = slave input.

The update equations were derived based on the backpropagation algorithm. The weights in the second layer of the master network are updated using equation 4.2.

$$V2M_{new} = V2M - \mu[2e\ (\underline{U}_S)\ (a^T)] \quad (4.2)$$

where μ = learning rate

$\underline{U}s$ = "extended" slave input

$a = \Gamma\{\Sigma [(V1M)\ (U_M)] + C1M\}$

$\Gamma(f)$ = sigmoid activation function

e = error = $(y - y_{desired})^2$

$\underline{U}s$ is a vector with a size of K+1 that consists of Us except the last element is 1, where K is the number of output nodes in the master network. In this case, K would be 2. The biases of the output nodes in the master network are updated using equation 4.3.

$$C2M_{new} = C2M - \mu[(2e)\ (\underline{U}_S)\ ] \quad (4.3)$$

The weights for the first layer in the master network are updated from equations 4.4 to 4.6.

$$V1M_{new} = (V1M1 + V1M2)\ /2 \quad (4.4)$$

where $$V1M1 = V1M - \mu[(2e)(Us)(V2M(j,1) \otimes a')](U_M) \quad (4.5)$$

$$V1M2 = V1M - \mu[(2e)(V2M(j,2) \otimes a')](U_M) \quad (4.6)$$

The symbol $\otimes$ is for element-wise matrix product where the weights in column j of V2M are multiplied by the element in row j of a', and then summed together. The biases of the hidden nodes in the master network are updated using equations 4.7 to 4.9.

$$C1M_{new} = (C1M1 + C1M2)\ /2 \quad (4.7)$$

where $C1M1 = C1M - \mu[(2e)(Us).\ (V2M(j,1) \otimes a')]$ (4.8)

$C1M2 = C1M - \mu[(2e)\ (V2M(j,2) \otimes a')]$ (4.9)

Detailed derivations of the update equations can be seen in Appendix B.

4.4.2 Nonlinear-nonlinear HSNN

For the nonlinear-nonlinear HSNN, both the master and the slave networks are non-linear, as shown in Figure 4.3. As seen in the figure, the slave network now has a hidden layer. The variable chosen as the slave input should have a direct, strong and non-linear relationship to the output variable compared to other input variables. The number of outputs of the master network is the same as the number of parameters of the slave network. Therefore, if there were three nodes in the hidden layer, the master network would have ten outputs. The following update equations are based on three hidden nodes in the slave network. The equations can be easily extended to have more or less hidden nodes. Detailed derivations are given in Appendix B.

In the update equations, it is important to note that the weights of the first layer of the slave network, V1S, corresponds to b_1, b_2 and b_3, which are the first three outputs of the master network. The bias of the hidden nodes, C1S, corresponds to b_4, b_5 and b_6, which are the fourth to sixth output of the master network. The weights of the second layer, V2S, corresponds to b_7, b_8 and b_9, which are the seventh to ninth outputs of the master network. Finally, the bias of the output node, C2S, corresponds to b_{10}, which is the last output of the master network.

The output of the nonlinear-nonlinear HSNN is calculated as in Equation 4.10.

$$y = \Sigma\,[(V2S)\ (g)] + C2S \qquad (4.10)$$

where $g = \Gamma[(V1S)\ (U_S) + C1S]$

The weights on connections ending at the first three outputs of the master network are updated using equation 4.11.

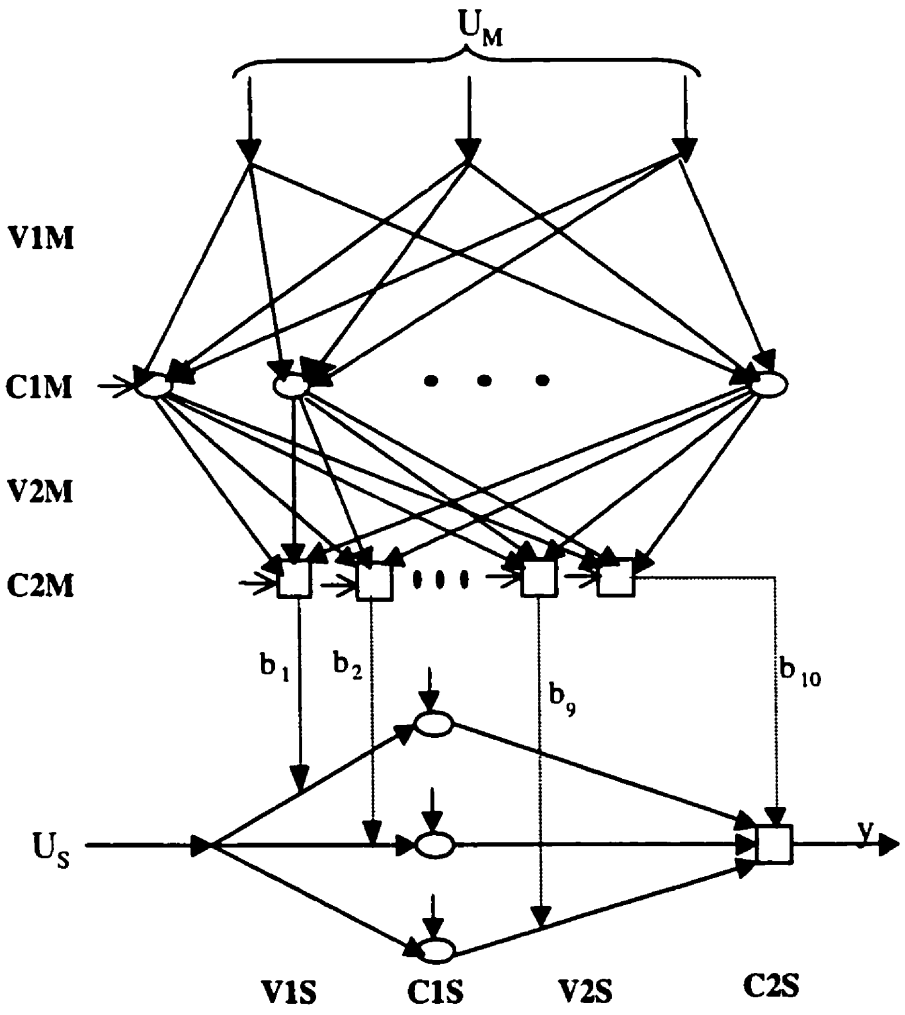

Figure 4.3: A nonlinear-nonlinear HSNN.

$$V2M_{new}(j, 1\text{-}3) = V2M(j, 1\text{-}3) - \mu[(2e)(U_S)(V2S \otimes g')\,(a)] \quad (4.11)$$

where $a = \Gamma\{\Sigma\,[(V1M)\,(U_M)] + C1M\}$

j = the subscript for all the nodes in the hidden layer of the master network.

The weights on connections ending at the fourth to sixth outputs of the master network are updated using equation 4.12.

$$V2M_{new}(j, 4\text{-}6) = V2M(j, 4\text{-}6) - \mu\,[(2e)\,(V2S \otimes g')\,(a)] \quad (4.12)$$

The weights on connections ending at the seventh to ninth outputs of the master network are updated using equation 4.13.

$$V2M_{new}(j, 7\text{-}9) = V2M(j, 7\text{-}9) - \mu[(2e)\,(g)\,(a)] \quad (4.13)$$

The weights on connections ending at the tenth output of the master network are updated using equation 4.14.

$$V2M_{new}(j, 10) = V2M(j, 10) - \mu[(2e).(a)] \quad (4.14)$$

The biases of the first three output nodes of the master network are updated using equation 4.15.

$$C2M_{new}(j, 1\text{-}3) = C2M\,(j, 1\text{-}3) - \mu\,[(2e)(U_S)\,(V2S \otimes g')] \quad (4.15)$$

The biases of the fourth to sixth output nodes of the master network are updated using equation 4.16.

$$C2M_{new}(j, 4\text{-}6) = C2M\,(j, 4\text{-}6) - \mu\,[(2e)\,(V2S \otimes g')] \quad (4.16)$$

The biases of the seventh to ninth output nodes of the master network are updated using equation 4.17.

$$C2M_{new}(j, 7\text{-}9) = C2M\,(j, 4\text{-}6) - \mu[(2e)(g)] \quad (4.17)$$

The bias of the tenth output node of the master network is updated using equation 4.18.

$$C2M_{new}(j, 10) = C2M\,(j, 10) - \mu[2e] \quad (4.18)$$

The update equations for the master network weights in the first layer are as follows:

$$V1M_{new} = (V1M1 + V1M2 + V1M3 + V1M4)/4 \quad (4.19)$$

where $V1M1 = V1M - \mu\,[(2e)\,(Us)(V2S \otimes g')\,(V2M(j,1\text{-}3) \otimes a')]\,(U_M)$ (4.20)

$$V1M2 = V1M - \mu\,[(2e)\,(V2S \otimes g')\,(V2M(j,4\text{-}6) \otimes a')](U_M) \quad (4.21)$$

$$V1M3 = V1M - \mu[(2e)\,(g)\,(V2M(j,7\text{-}9) \otimes a')](U_M) \quad (4.22)$$

$$V1M4 = V1M - \mu[(2e)\,(V2M(j,10) \otimes a')](U_M) \quad (4.22)$$

The update equations for the master network biases in the first layer are as follows:

$$C1M_{new} = (C1M1 + C1M2 + C1M3 + C1M4)/4 \quad (4.23)$$

where $C1M1 = C1M - \mu[(2e)(Us)(V2S \otimes g')\,(V2M(j,1\text{-}3) \otimes a')]$ (4.23)

$$C1M2 = C1M - \mu[(2e)\,(V2S \otimes g')\,(V2M(j,4\text{-}6) \otimes a')] \quad (4.24)$$

$$C1M3 = C1M - \mu[(2e)\,(g)(V2M(j,7\text{-}9) \otimes a')] \quad (4.25)$$

$$C1M4 = C1M - \mu[(2e)\,(V2M(j,10) \otimes a')] \quad (4.26)$$

4.4.3 Output-tuned HSNN

The output-tuned HSNN has a different structure than the completely driven HSNN. Figure 4.4 illustrates a schematic diagram of an output-tuned HSNN. The master network has two outputs, which is a gain, b_1, and a bias, b_2, for the output of the slave network. This type of network is recommended for zeroing the network output, y, at certain values of an input variable; in this case, the input variable which causes the output to be zero is the master input, U_M. Output-tuned HSNN is also recommended when there is a constraint to be met; in this case, the output of the whole constraint equation is U_M.

The output of the output-tuned HSNN is calculated as in Equation 4.27.

$$y = (y_s)\,(b_1) + b_2 \quad (4.27)$$

where $y_s = \Sigma\,[(V2S)\,(g)] + C2S$

$$g = \Gamma\{[(V1S)\,(U_S)] + C1S\}$$

The output of the master network, B, is:

$$B = [b1\ \ b2]' = \Sigma\,[(V2M)\,(a)] + C2M \quad (4.28)$$

where $a = \Gamma\{[(V1M)\,(U_M)] + C1M\}$.

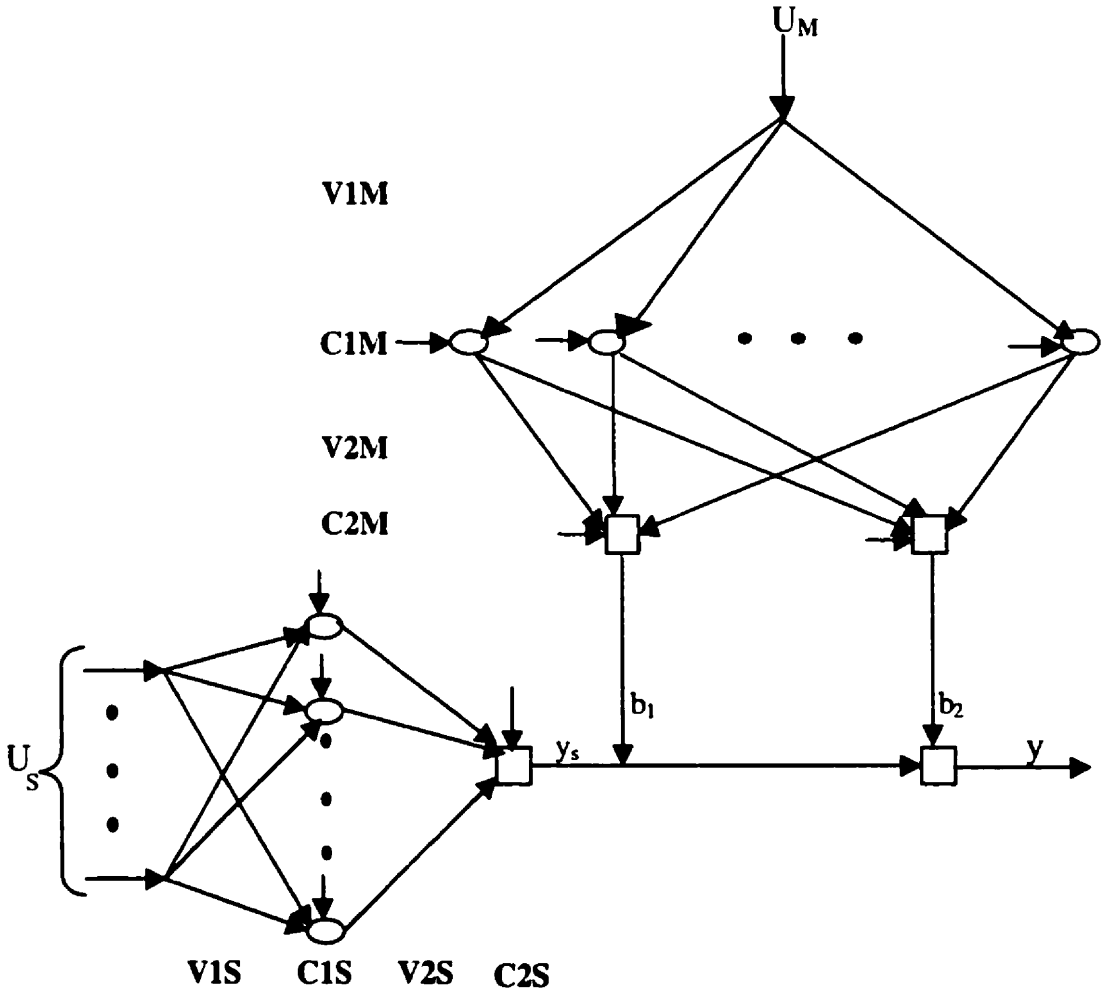

Figure 4.4: Output-tuned HSNN.

The weights on connections ending at the first output of the master network are updated using equation 4.29.

$$V2M_{new}(j, 1) = V2M(j, 1) - \mu[(2e)(y_s)(a)] \quad (4.29)$$

where j = the subscript for all the nodes in the hidden layer of the master network.

The weights on connections ending at the second output of the master network are updated using equation 4.30.

$$V2M_{new}(j, 2) = V2M(j, 2) - \mu[(2e)(a)] \quad (4.30)$$

The bias of the first output node of the master network is updated using equation 4.31.

$$C2M_{new}(j, 1) = C2M(j, 1) - \mu[(2e)(y_s)] \quad (4.31)$$

The bias of the second output node of the master network is updated using equation 4.32.

$$C2M_{new}(j, 2) = C2M(j, 2) - \mu[2e] \quad (4.32)$$

The update equations for the master network weights in the first layer are as follows:

$$V1M_{new} = (V1M1 + V1M2)/2 \quad (4.33)$$

where $$V1M1 = V1M - \mu[(2e)(y_s)(V2M(j,1) \otimes a')](U_M) \quad (4.34)$$

$$V1M2 = V1M - \mu[(2e)(V2M(j,2) \otimes a')](U_M) \quad (4.35)$$

The update equations for the master network biases in the first layer are as follows:

$$C1M_{new} = (C1M1 + C1M2)/2 \quad (4.36)$$

where $$C1M1 = C1M - \mu[(2e)(y_s)(V2M(j,1) \otimes a')] \quad (4.37)$$

$$C1M2 = C1M - \mu[(2e)(V2M(j,2) \otimes a')] \quad (4.38)$$

Unlike the completely driven HSNN, the slave network parameters must also be calculated with update equations. The weights in the second layer of the slave network are updated using equation 4.39.

$$V2S_{new} = V2S - \mu[(2e)(b_1)(g)] \quad (4.39)$$

The bias of the output node of the slave network is updated using equation (4.40).

$$C2S_{new} = C2S - \mu[(2e)(b_1)] \quad (4.40)$$

The weights in the first layer of the slave network are updated using equation 4.41.

$$V1S_{new} = V1S - \mu[(2e)(b_1)\ (V2S \otimes g')]U_S \quad (4.41)$$

The biases of the first layer of the slave network are updated using equation (4.42).

$$C1S_{new} = C1S - \mu[(2e)(b_1)\ (V2S \otimes g')] \quad (4.42)$$

4.5 SERIAL NETWORK MODELS

Two types of serial network models were developed:

- serial RBFN-RBFN
- serial RBFN-RBFN- output-tuned HSNN

Figure 4.5 illustrates the general structure of the serial RBFN-RBFN model, where I represents the input variables and y represents the output variables. Referring to Figure 4.5, Network 1 may have 1 to 3 outputs. Network 1, a RBFN, was determined from the previously performed single network runs. The inputs of Network 2 are the original inputs, I, and the outputs, O, of Network 1. Network 2 is a RBFN.

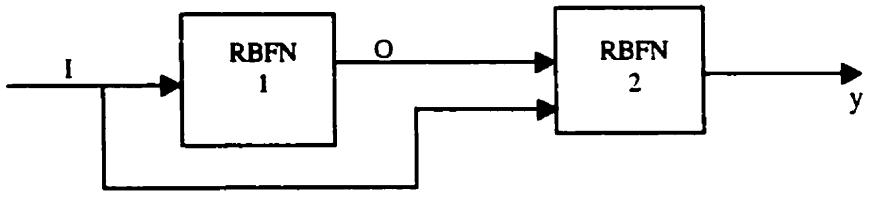

Figure 4.5: Serial network model.

According to the definition of grey box models used in this thesis, the serial network model is considered a grey box model because the selections of the intermediate variables, O, are based on prior knowledge. Intermediate variables are variables that are easier for the network to predict than the final output variable. At the same time, the intermediate variables are strong functions of the final output, and can therefore provide more information for Network 2 to better predict the final output variable.

In serial RBFN-RBFN - output-tuned HSNN models, output-tuned HSNN is used in series with a serial RBFN-RBFN model, as shown in Figure 4.6. Two types of constraints were tried in different models for the prediction of y for the M-W flash system. The first is:

$$y = 0 \text{ at } \quad V = 0 \tag{4.43}$$

A second constraint is from the mass and component balances:

$$y = 0 \text{ at } \quad [(F)(z) - (L)(x)] / V - y_i = 0 \tag{4.44}$$

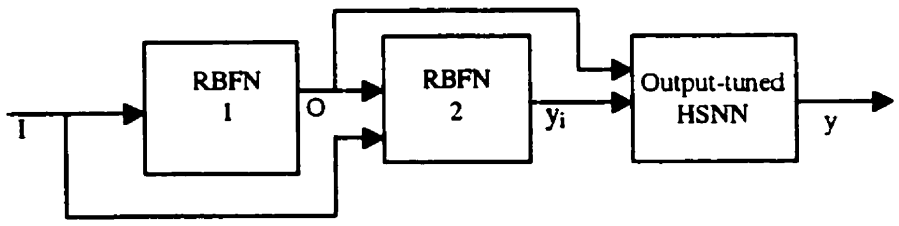

Figure 4.6: Serial RBFN - RBFN - output-tuned HSNN.

4.6 HYBRID NETWORK MODELS

In the hybrid network, network models are coupled with first principles models or mechanistic equations. There are undoubtedly many ways to do this. However, in this work, since the models are going to be used for RTO, the mechanistic models incorporated must be simple and straightforward to solve. Similarly, although previously introduced models may seem to be more difficult to train compared to a standard ANN model, once trained, all of them can be easily solved like any standard ANN model.

The hybrid models can be divided into two types. The first type is the serial network - FP model, much like the serial semi-mechanistic model. The only difference here is that the network model output is not a process parameter, but a process output variable. Two different models were tested in this category. Model 1 (Figure 4.7) uses FPM 1, which is the equilibrium relation:

$$y = (K)\,(x) \tag{4.43}$$

where y = vapour composition of methanol in stream V,

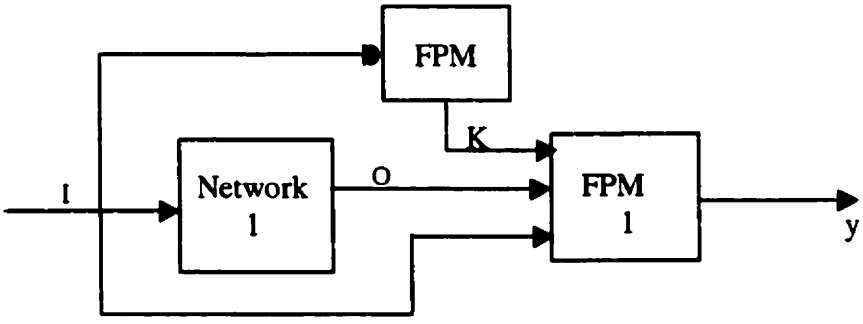

Figure 4.7: Type 1 serial hybrid model with K value (Model 1).

K = equilibrium constant,

x = liquid composition of methanol in stream L.

The value for K was obtained from Aspen Plus at the corresponding state, but it is also possible to calculate K from correlations. Another option would be to have a separate network to correlate K to the corresponding input conditions.

Model 2 for Type 1 category (Figure 4.8) uses FPM 2, the component balance:

$$y = [(F)(z) - (L)(x)] / V \qquad (4.44)$$

For this hybrid model to be used, the intermediate output, O, must be both L and x or V and x, or V, L and x. Conditional statements were added in all hybrid models to filter out negative values from intermediate variables predicted by Network 1. An additional condition was also added in the models that used Equation 4.44 to set y at zero if V is less than or equal to zero. These conditions were added not because of the process or the inaccuracy of the specific model structure, but because an inherent weakness of all standard ANN models for function mapping is that the models are not able to provide exact zeroing.

Type 2 of the hybrid model is similar to Type 1, except models in this category have an additional network at the end of the model. Therefore, there are three levels in Type 2 models. The network in the third level is a RBFN.

Four different models were developed for this category. Models 1 and 2 both used FPM 1 and had very similar configuration, which is shown in Figure 4.9. The only difference is that in addition to the model input, I, and the output of FPM 1, Model 1 had the intermediate output, O, going to the network in the third level as an input. The structure of Model 3 is shown in Figure 4.10. This model used FPM 2 in the second level. The structure of Model 4 is shown in Figure 4.11. This model used both FPM 1 and FPM 2 in the second level.

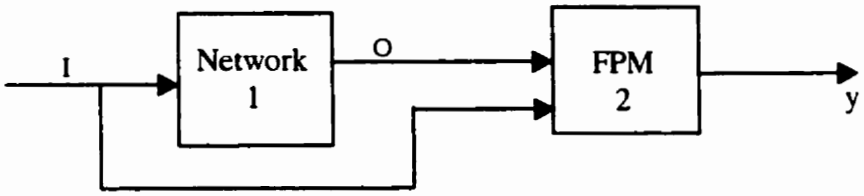

Figure 4.8: Type 1 serial hybrid model with component balance (Model 2).

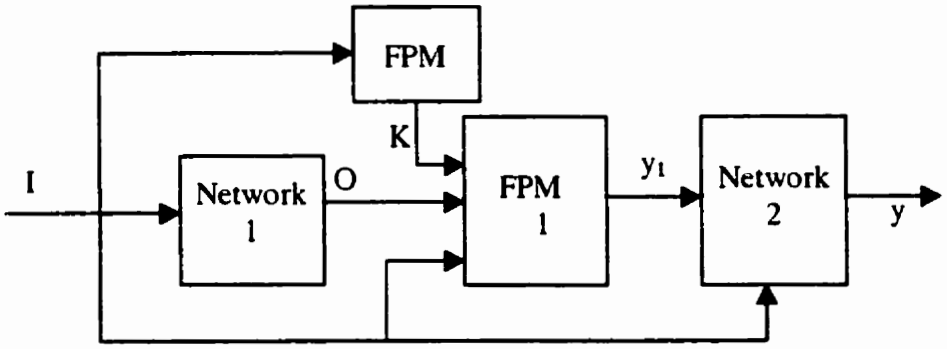

Figure 4.9: Type 2 hybrid model (Model 2).

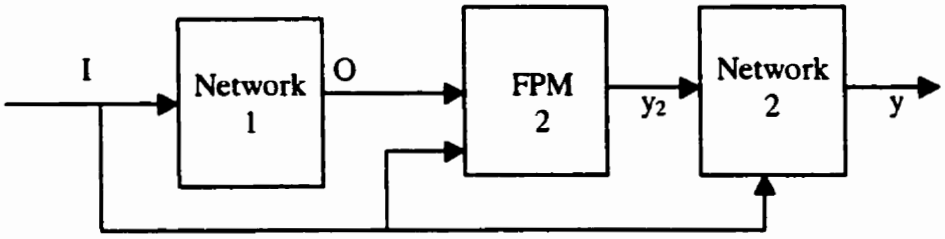

Figure 4.10: Type 2 hybrid model (Model 3).

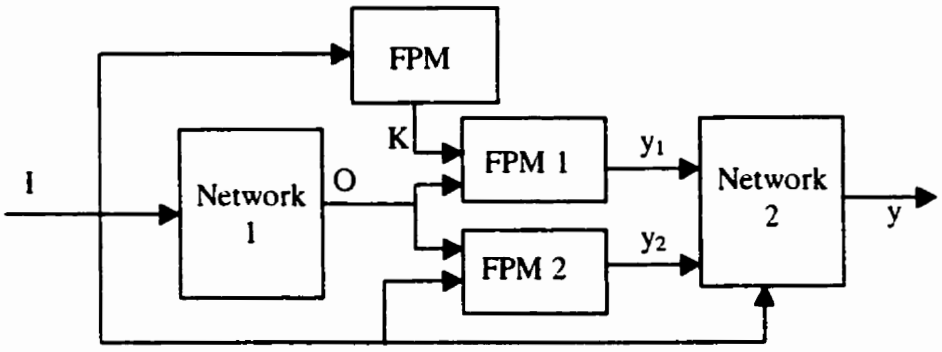

Figure 4.11: Type 3 hybrid model (Model 4).

4.7 SUMMARY

In this chapter, detailed descriptions of the models developed are presented. Brief descriptions of the algorithms and structure of the models are included. Update equations for the three HSNN-type models are also given.

Chapter 5 presents the results of the different models described in this chapter. The models were tested on the flash systems and the crude oil distillation column. A comparison between the models and between the two different chemical processes is also discussed.

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 OVERVIEW

The models described in Chapter 4 were developed and simulated under the MATLAB environment. The numerical results obtained are presented here.

Two standard ANN models, multi-layer perceptrons (MLP) and radial basis function networks were tested first to determine a base case for comparison. Models were developed for the methanol-water (M-W) system, which is a thermodynamically non-ideal mixture.

More complex model structures were used for variables that were difficult to predict. A simple model that could sufficiently predict output variables is preferred. More complex models were tested only if a variable was unsatisfactorily predicted by standard ANN models as these models were more difficult to develop and train. For example, developing HSNN models are more involved because changes in the number of variables and in certain model structures require more adjustment and tuning compared to a standard ANN model. HSNN models also require longer training times because of the complexity of the model structure and the backpropagation algorithm used to train the network which was used by Bittanti and Savaresi (1998). In addition, as a consequence of its architecture, HSNN models are only suitable for single output

variable prediction. Therefore, it is not practical to develop more complex models for all variables when simpler models are sufficient.

Results for the different models are presented in the following order:

- standard MLP and RBFN models,
- completely driven HSNN models,
- serial network models,
- hybrid network models, and
- crude oil distillation tower models.

For the M-W system, all models are trained with the same batch of 150 training data and 70 testing data, which are described earlier in Chapter 3. The standard ANN models were initially tested with different numbers of training data ranging from 70 to 400, and the data set with 150 points was found to give the best results in terms of prediction and training time.

To determine the number of training data points, both the RBFN and MLP models were trained with different training data sets. Both models used the same maximum acceptable sum of squared error (MSSE) during training and the default number of allowable of iterations in MATLAB, which is 10,000. As described in Chapter 4, once the spread constant, σ, and MSSE were specified, the OLS algorithm used to train the RBFN automatically selects the centers, and thus the number of nodes for the model. For MLP, different numbers of hidden layers and nodes were tested.

For example, Table 5.1 shows the prediction of V with 150 and 300 training data points. In the table, "CPU s" represents the time in CPU seconds taken for training the network on a personal computer with a Pentium 200 MHz micro-processor. The RMS error for the RBFN model trained with 300 training data points was slightly lower than the RMS error trained with 150 data points. However, the training time when 300 data points were used was almost ten times more than when 150 data points were used. Similarly, the MLP network trained with the higher number of training data gave better

predictions. The structure with two hidden layers (30 and 15 nodes) gave better predictions than the structure with a single hidden layer (30 nodes). Unfortunately, the two-hidden layer structure trained with 300 data points took such a long time to train that it is not practical to use for other variables as well. Therefore, the 150 data points were chosen as the training set.

The training and testing data for the crude distillation tower depends on the different sections of the column, and as described in the crude tower results section in Chapter 3. Similar to the M-W system, different sets of training data were tested to find the training set that gave the best results.

Evaluations of the models are based on root mean squared (RMS) error for each model prediction. Error is defined as the difference between desired (or actual value provided by the testing data) output value and the predicted output value. Training time will also be taken into consideration, mainly because of the convenience in developing models with short training cycle. The training time will only be a major concern when the model is periodically updated on-line. For all the models, the results presented in this chapter are the best ones obtained after numerous trials of different training error tolerance, spread constant, number of nodes and layers, and learning rates, whichever parameters that were applicable for the different models.

Table 5.1: Prediction of V using 150 and 300 training data points.

	RBFN		MLP (1 layer)		MLP (2 layers)	
No. of data	150	300	150	300	150	300
RMS error	0.0360	0.0313	0.0565	0.0443	0.0434	0.0380
CPU s	26.4	244	415	833	772	16,790

5.2 STANDARD ANN

5.2.1 Base Case Model Selection

There are three types of standard ANN models: two MLP and one RBFN. All the models have T, P, F and z as the input variables with the same training and testing data. The models also use the same MSSE and the default maximum number of iteration to avoid long training times. The MLP models were developed with two different training algorithms, which were backpropagation (BP) with variable learning rate, μ, and Levenberg-Marquardt (L-M) training. Models with one and two hidden layers were tested. The number of nodes in each hidden layer was also varied. The best MLP models for both training methods had two hidden layers with 30 and 15 nodes in the first and the second hidden layers respectively. For the RBFN model, the OLS algorithm used in MATLAB iteratively selected the centers (or nodes) that yielded the least error. Therefore, only the MSSE and the spread constant had to be changed to find the best RBFN model.

Table 5.2 shows the RMS error of the test data prediction for each output variable of the M-W flash system. As seen in Table 5.2, RBFN gave the smallest RMS errors for all the variables. The predictions of both MLP models yielded similar results for all output variables, except for q where the MLP trained with the L-M algorithm had an RMS error that is about half of the RMS error of the MLP trained with BP. The model trained with the L-M algorithm predicted three out of five variables better than the model trained with BP. Both models use a random initialisation algorithm for the weights and biases; this initial value could also influence the model prediction. The training times for the RBFN models are ten to twenty times shorter than for the MLP models. Comparing the two MLP models, the L-M algorithm had three to five times shorter training times than the BP algorithm with variable μ. Based on these results, RBFN were taken as the base case for comparison with other models.

Table 5.2: Comparison of the best results obtained with RBFN and two MLP models.

Output	RBFN model		MLP w/ BP & var. μ		MLP with L-M	
	RMS Error	CPU s	RMS Error	CPU s	RMS Error	CPU s
V	0.0360	26.41	0.0434	772.0	0.0572	185.9
y	0.1119	12.63	0.2421	852.3	0.2581	318.4
L	0.0267	11.36	0.0321	809.5	0.0314	237.4
x	0.0309	19.38	0.0427	822.0	0.0410	251.1
q	0.0325	21.59	0.0702	614.3	0.0357	279.6

Since all models are compared against the base case, model predictions that yielded an improvement relative to the base case are deemed as satisfactory. No standard error limits are found in the literature, mainly because acceptable or satisfactory predictions are somewhat subjective, depending on the process and the objective of the model.

5.2.2 RBFN Models for M-W System

Table 5.3 provides the complete results of the standard RBF models for the M-W system. The first section of the table gives the RMS error of each output variable that was individually predicted. The second section gives the RMS error when two output variables were predicted together with a single RBFN. The third, fourth and fifth sections give the RMS error when three, four and five output variables were predicted together respectively. MSSE listed in the second column is the maximum allowable sum of squared error for the training set (that is, during identification). Total RMS errors in the third column of Table 5.3 is the sum of the RMS error of all the output variables of the model. Individual RMS error, in the third column, is the RMS error of

Table 5.3: Results of RBF network models.

Outputs (y1,y2,y3,y4,y5)	MSSE	Total RMS Error	Individual RMS Error
V	0.05	0.0360	
y	0.15	0.1119	
L	0.01	0.0267	
x	0.01	0.0309	
q	0.05	0.0376	
V, y	0.01	0.1478	0.0354, 0.1124
V, x	0.03	0.0599	0.0297, 0.0302
V, L	0.01	0.0528	0.0264, 0.0264
V, q	0.10	0.0533	0.0290, 0.0243
y, x	0.10	0.1625	0.1324, 0.0301
V, y, x	0.20	0.1782	0.0434, 0.1032, 0.0316
V, y, L	0.03	0.1666	0.0267, 0.1132, 0.0267
V, L, x	0.01	0.0828	0.0257, 0.0257, 0.0314
V, x, q	0.01	0.1047	0.0498, 0.0313, 0.0236
V, y, L, x	0.10	0.1993	0.0279, 0.1123 0.0279, 0.0312
V, y, L, x, q	0.03	0.2188	0.0269, 0.1134, 0.0269, 0.0311, 0.0205

each variable in the respective order given in the first column. For the purpose of simplicity, plots of the output variables are not shown here because there are too many variables and combinations of variables predicted. However, plots are shown in a later section that discusses comparisons of the results obtained.

From Table 5.3, although the total RMS error may seem to increase as the number of output variables are increased, the individual RMS error showed otherwise for most of the variables. In fact, suitable variable combinations are found to decrease the RMS error of most of the variables. For example, when only V was predicted, the RMS error was 0.0360; however, the RMS error of V when predicted in the output variables combination of [V, x], and [V, L, x] were 0.0297 and 0.0257 respectively. The same trend in prediction error could be seen with the rest of the variables, except for x and y,

the composition of methanol in the liquid and vapour streams respectively. The RMS errors of x and y in different output combinations are almost the same. However, the results obtained using RBFN show that y is the most difficult variable to predict.

The results in Table 5.3 are the best obtained using RBF networks, after trying several different spread constants, σ, and the maximum allowable sum of squared error, MSSE. Determining the suitable σ for the RBFN models was fairly easy, as σ for the same data would be almost the same even for different variable predictions since the data were normalised. For all the RBFN models developed, σ was either 0.15 or 0.20. The MSSE, however, varied depending on the behaviour of the variable being modelled. The variables that had good predictions could be modelled tightly, with small MSSE. For example, for the predictions of V, L, x and q, shown in Table 5.3, the MSSE are 0.05, 0.01, 0.01 and 0.05 respectively. For the prediction of y, however, the MSSE is 0.15. This is because y is not only highly non-linear, but also discontinuous. A tightly tuned model for y will not be able to generalise well because the surface for convergence has been strictly limited by the requirements of a small MSSE.

5.2.3 RBFN Models for B-T System

To ensure that the unsatisfactory result obtained for the prediction of y was not due to the non-ideal behaviour of the M-W mixture, the same output variables were predicted with a benzene-toluene (B-T) flash system, which is thermodynamically ideal. As in the M-W system, the range modelled covered the single-phase liquid, two-phase and single-phase vapour regions. The training and testing data were generated in Aspen Plus by varying T (88-100 °C), P (0.85-1.05 bar), F (70-170 kmol/hour), and z (0.28-0.71). Since a B-T mixture is thermodynamically ideal, the results would indicate if the

non-ideality of the M-W mixture were a major factor in the difficulty of predicting the system.

From the results given in Table 5.4, it can be seen that the RMS errors for all output variables are only slightly lower that those of the M-W system, except for L and x. The prediction for the vapour composition of benzene, y, is still not satisfactory. After testing with the B-T system, it can be concluded that y for any flash system that covers single-phase liquid, single-phase vapour and two-phase vapour-liquid regions would be difficult to predict. Therefore, a more complex structure is needed to get a better prediction of y in this range of data.

5.2.4 RBFN Models for Two-Phase Region

To determine that the RBF network is indeed able to model the flash system in the two-phase region, all zeroes and ones in the original pool of training and testing data generated from the M-W Aspen Plus simulation were discarded. The data that are left are therefore points in the two-phase region, where both liquid and vapour are present. 100 data points from the original training data and 70 data points from the original testing data were then selected at approximately equal intervals. The 100 training data

Table 5.4: Results of the B-T system using RBFN models.

Variables	M-W system	B-T system
V	0.0360	0.0264
y	0.1119	0.1034
L	0.0267	0.0270
x	0.0309	0.0612
q	0.0325	0.0160

set was found to be the best after trying several different numbers of training data. This two-phase region is actually the practical operating region for flash systems.

The results of the model predictions are shown in Table 5.5. As seen in the table, the results obtained are much better than for the model that spans the single and two-phase regions. The predictions are very good especially for y, x and q, where the RMS errors were an order of magnitude lower than the predictions of the base case. V and L, however, had only slight improvements; this is most probably because the RBFN was able to distinguish the linear relationship between F and V and L even in the original data range. Consequently, the improvement seen in the prediction of V and L are not as substantial as for the three other output variables. The good results obtained proves that the RBF network is indeed able to model the non-linearity of the system, but is unable to account for the discontinuity at the edge of phase envelope where the mixture is in the single-phase region. Fortunately, in practical industrial applications, the operation of a flash system is limited within the two-phase region. Nevertheless, if a wider region was desired, a more complex model should be used, especially to predict y.

Table 5.5: Results of standard RBFN model for M-W system in the two-phase region.

Outputs	M-W system (Base case)	M-W system (2-phase only)
V	0.0360	0.0114
y	0.1119	0.0065
L	0.0267	0.0108
x	0.0309	0.0074
q	0.0325	0.0095

5.2.5 Predicting y

The problem in predicting y stems from the abrupt change in composition from the two-phase region to the single-phase liquid region. There may be conditions that can be given to overcome this problem; however, since this is a test case, a generalised approach is preferred because similar problems may also occur in other processes. For example, Yang et al. [1999] reported that straightforward standard ANN was not able to model sharp changes in monomer and initiator concentrations for styrene polymerisation. In addition, simple conditions like setting y to zero when V is zero cannot work because of the inherent weakness of all standard ANN models that cannot give exact zeroing for function mapping. Figure 5.1 shows the plot of normalised actual V versus normalised predicted V. Out of six points where the actual y were zero, four were predicted to be slightly positive (about 0.06) and two were predicted to be slightly

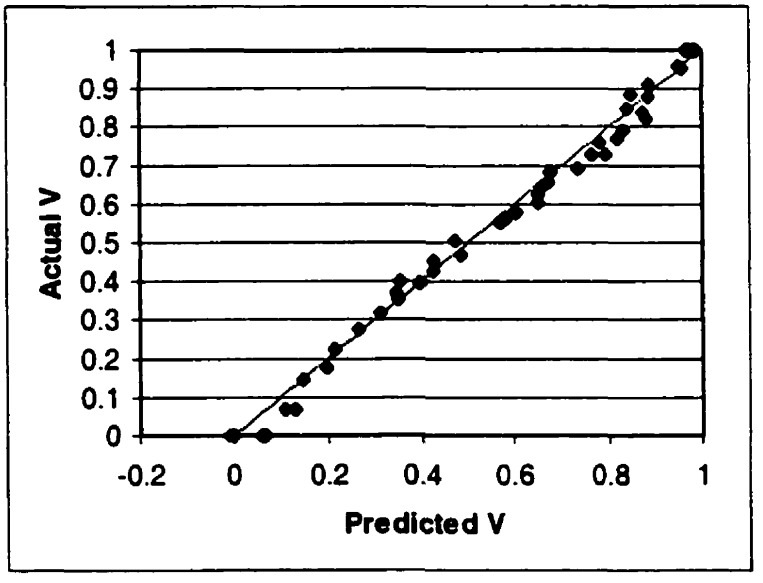

Figure 5.1: Plot of normalised actual V versus normalised predicted V.

negative (about -0.02). Consequently, the prediction of V cannot be directly used as a condition.

Therefore, to have a generalised approach in handling this problem, more complex network models must be used. The following section discusses the results of more complex ANN models in predicting y.

5.3 COMPLETELY DRIVEN HSNN

Bittanti and Savaresi [1998] reported that completely driven Hierarchically Structured Neural Networks (HSNN) was able to handle discontinuities, especially when the discontinuity depends on a specific input variable. HSNN would therefore be a suitable network model to test. Two types of completely driven HSNN models were developed for the prediction of y: linear-nonlinear HSNN and nonlinear-nonlinear HSNN. Output-tuned HSNN was not developed for predicting y as a single model on its own because no clear-cut constraint could be given from the input variables. This type of HSNN, however, could be used in the serial model, which will be described in a later section.

Both models use the same training and testing data file for the M-W system as in the base case. Similar to the base case, the inputs to the HSNN models are also T, P, F and z. However, for HSNN, the input variables are further divided into slave and master inputs.

5.3.1 Linear-nonlinear HSNN

Figure 4.2 illustrates the best linear-nonlinear HSNN structure used for prediction for the M-W system. The master network was kept as a single hidden layer, but the number of nodes was varied to find the best model. The lowest RMS error is found when the

master network has six hidden nodes and two output nodes. The outputs of the master network provided values for the weight and bias of the slave network. As mentioned in Chapter 4, for linear-nonlinear HSNN, the slave input should have a direct influence and is approximately linear to the output of the HSNN. The input variables to the network are T, P, F and z. If F was the slave input, then the three remaining input variables were the master input. Different slave input and number of hidden nodes for the master network were tested to find the best configuration. The best configuration found had six nodes in the hidden layer of the master network.

Table 5.6 gives the best results obtained for single output predictions with HSNN and standard RBFN. The second column of the table gives the slave input variable used in the linear-nonlinear HSNN model. For example, F was the slave input for the prediction of V, and four different slave inputs, z, F, K and T were individually tested for the prediction of y. Two different K-values data were used. In the first model, the K-values being used are taken directly from Aspen Plus. Aspen Plus estimates K-values from physical property methods, without taking into account whether both liquid and vapour phases are actually present at a particular state. Therefore, the K-values are incorrect when the M-W mixtures only exist in a single phase because equilibrium constants are valid only within the two-phase region. The K-values used in the second model are modified to indicate when there is only a single vapour or liquid phase.

Referring to Table 5.6, among the three output variables predicted, V had the lowest RMS error. The result for V is also better than the base case. This is an expected outcome because from mass balance, since V has a linear relationship with F. The RMS errors of the first three models used to predict y were close. Even though z and F are not linearly related to y, both linear-nonlinear HSNN models are able to give a better prediction of y than standard RBFN. When F was used as the slave input to predict y, the RMS error obtained is only slightly higher than when z was used as the slave input. This is reasonable because from mass and component balances, F and z are both strong, but slightly non-linear functions of y. Using K-values with errors as the

Table 5.6: Results of linear-nonlinear HSNN models.

Output	Slave Input	MSSE	RMS Error for Standard RBFN	RMS Error for Linear-nonlinear HSNN
V	F	0.03	0.0360	0.0240
y	z	0.07	0.1119	0.0893
	F	0.07		0.0885
	K(Error in K)	0.07		0.0857
	K (correct K)	0.04		0.0344
	T	0.16		0.1593
x	z	0.027	0.0325	0.0362
	F	0.035		0.0311

slave input yielded similar results as the first two models because the errors in the K-values would give misleading information in the single-phase region. As expected, the best result in predicting y was obtained when the correct K-values were used as the slave input. When T was the slave input, the model prediction is worse than the prediction of the RBFN model. This has most probably occurred because the relationship between y and T is not only highly non-linear, but is also not direct one. As seen in Chapter 3, T influences the value of y through the equilibrium constant K.

The RMS errors using RBFN and linear-nonlinear HSNN for the prediction of x, are close. None of the input variables were suitable as a slave input for predicting x; F was found to be the best choice. Using z as a slave input yielded a RMS error that is also slightly higher than when F was used as the slave input. Therefore, when a suitable slave input is not available, using a simple standard RBFN model is sufficient. Also, simple models that can yield acceptable predictions are better because of the short training time for developing the model. The training times for predicting y with z and F

as the slave input are 29,826 CPU seconds and 21,504 CPU seconds respectively, which are much greater than the training time of about 13 CPU seconds for the base case.

5.3.2 Nonlinear-nonlinear HSNN

The structure for the nonlinear-nonlinear HSNN model is illustrated in Figure 4.3. Both the master and slave networks have one hidden layer. The best model developed has 20 nodes in the hidden layer and 10 nodes in the output layer of the master network. The master network outputs are the parameters of the slave network, which has three nodes in the hidden layer. The variable chosen as the slave input can be non-linear, but should also be directly a function of the output variable. The complexity of nonlinear-nonlinear HSNN makes it suitable for modelling difficult-to-predict variables like y. Other output variables are not modelled because the results obtained with simpler models are deemed to be satisfactory.

From the results of the nonlinear-nonlinear HSNN shown in Table 5.7, the RMS error for both models are less than that using standard RBFN, but the training times were much longer mainly because of the complex structure and the backpropagation algorithm. The training times for the two models with F and z as the slave inputs are 28,790 CPU seconds and 10,681 CPU seconds respectively. Nevertheless, once training is completed and the model is obtained, the execution of the model is very fast.

Table 5.7: Results of nonlinear-nonlinear HSNN models.

Output	Slave Input	MSSE	RMS Error for Nonlinear-nonlinear HSNN	Training Time (CPU s)
y	F	0.05	0.0884	28,790
y	z	0.06	0.0820	10,681

5.4 SERIAL NETWORK MODELS

All the serial network models basically have the same structure as shown in Figure 4.5. The first network for all the serial models is an RBF network that is used to calculate intermediate output variables that had good predictions, based on the results shown in Table 5.3. All models were used to predict y as the final output. Two different types of networks were tested as the second network model: RBFN and output-tuned HSNN.

5.4.1 Serial RBFN-RBFN Models

In developing models in this category, several different intermediate variables were tested. Since the σ and MSSE for Network 1 were already known, only the parameters for Network 2 need to be systematically searched. The σ used for Network 2 for the different models ranged from 0.15 to 0.3. The MSSE for Network 2 for the prediction of y range from 0.05 to 0.30.

The results of the serial RBFN-RBFN models are listed in Table 5.8. Referring to the first column in Table 5.8, x→y indicates that the output of Network 1 is x, which is then included with the original input variables, I, as an input to Network 2 to predict y, since it is the most difficult variable to predict using standard networks. The third column in Table 5.8 provides the total RMS error for the prediction of all variables from the serial model, while the fourth column provides the individual RMS error of the first and second networks. Note that although the total error may seem large, the actual RMS error of each individual variable is not. The individual RMS error of the intermediate variables can be seen in Table 5.3.

The RMS error for the prediction of y, shown in column 4of Table 5.8 (RMS error for ANN 2), all gave slightly better results than prediction with a single RBFN, except for V,L → y. The additional information provided by the intermediate variables for the

Table 5.8: Results of serial RBF network models.

Output	MSSE (ANN 1/ANN 2)	Total RMS Error	RMS Error (ANN 1/ANN 2)
x → y	0.01/0.15	0.1397	0.0329/0.1067
V, L → y	0.01/0.05	0.1368	0.0804/0.1123
V, L, x → y	0.01/0.30	0.1392	0.0983 /0.1031
V, x → y	0.03/0.15	0.1172	0.0620/0.1088
V, x, q → y	0.01/0.10	0.1455	0.0835/0.1090

prediction of y allowed tighter training. Among the runs performed for the serial RBF networks, the runs with V,L,x → y and V,x → y gave satisfactory results, with both yielding the RMS error for prediction of y at 0.1031 and 0.1088 respectively. The variables involved in predicting y in both groups are strong functions of y. These are the variables needed in a material balance to solve for y.

5.4.2 Serial RBFN - RBFN - Output-tuned HSNN

Bittanti and Savaresi [1998] recommended that the output-tuned HSNN be used for implementing constraints. Referring to the structure of the output-tuned HSNN shown in Figure 4.4, an intermediate output is predicted by the slave network and the constraint is met by the master network. To predict y, the output-tuned HSNN is not used on its own because from previously tested models, it can be deduced that the slave network, which is actually an MLP, will not be able to properly predict y. In addition, the input variables available are not suitable as constraints, and there is not enough information to use the mass and component balance. Two output-tuned HSNN were developed. As presented in Chapter 4, the master input of the first model was V and the second was the component balance.

For all the models developed, the master network has six hidden nodes and one output node. The slave network has one hidden layer, but the number of nodes varies from one model to another. The master network has only one output, which is the gain that is multiplied to the output of the slave network. There is no bias added to the output of the slave network. From the paper by Bittanti and Savaresi [1998], the bias is normally used to set the output of the network to a specific constant value when a constraint is met. For the first constraint, when V is zero, y is also zero, and thus eliminates the need for a bias. For the second constraint, the value of y varies according to the constraint, and thus would also eliminate the need for a bias. Nevertheless, there were several models developed with a bias. The results were very poor, with the RMS errors for the prediction of y being greater than 0.3 for all models. Therefore, the rest of the models were developed with one output for the master network.

Table 5.9 presents the results of the serial RBFN-RBFN-output-tuned HSNN model. All models have the same intermediate outputs. The serial RBFN-RBFN model was chosen to have V, L and x as the intermediate outputs because this model has the best prediction of y_i (RMS error 0.1031) and can also be used to estimate the component balance. Referring to Table 5.8, the second and third columns list the master and slave inputs respectively. RMS errors of all models are lower than the serial RBFN-RBFN models. This shows that the output-tuned RBFN were able to act as a filter and implement the constraints to improve the prediction of y.

Table 5.9: Results of serial RBFN - RBFN - output-tuned HSNN models.

Intermediate Output	Master Input	Slave Input	MSSE of HSNN	Hidden nodes in slave	Final output, y, RMS error
V, L, x → y_i	V	V, y_i, L, x	0.02	12	0.0114
V, L, x → y_i	Eq. 4.44	V, y_i, L, x	0.04	18	0.0074

From Table 5.9, there was an improvement in the prediction of y between the first and the second models. The first model used the constraint in Equation 4.43, while the second model used the constraint in Equation 4.44. The RMS error of the second model, 0.0074, is the lowest one so far. The training time, at 212 CPU seconds, is also acceptable. Predictions of y from the models shown in Table 5.9 are all highly satisfactory compared to other models developed.

5.5 HYBRID NETWORK MODELS

There are two types of hybrid network models: hybrid RBFN-FPM (Type 1) and hybrid RBFN-FPM-RBFN (Type 2). Recalling the definition given in Chapter 4, FPM1 is the equilibrium relations equation (Equation 4.43) and FPM2 is the component balance equation (Equation 4.44). The hybrid models are arranged in a series of levels as discussed in Chapter 4.

5.5.1 Hybrid RBFN-FPM

There are two different kinds of hybrid RBFN-FPM models. As described in Chapter 4, Model 1 uses FPM 1 and Model 2 uses FPM 2. Figures 4.7 and 4.8 illustrate the schematic of Model 1 and Model 2 respectively. In this type of model, the output variable prediction comes directly from a FPM.

Table 5.10 presents the results obtained from Type 1 models. All models in Table 5.10 have V, L, and x as the intermediate outputs. In these models, the values of V and L are normalised so that the sum of V and L is F. FPM 1 is used to calculate the output variable, y, for both the first and second model; similar to the linear-nonlinear HSNN models, the difference in these two models are the K-values being used to calculate y

Table 5.10: Results of Type 1 hybrid structure.

Intermediate Output	Model	Total RMS	RMS (ANN 1/FPM)
V, L, x	1(error in K-values)	0.2263	0.0983 / 0.1280
V, L, x	1(correct K-values)	0.1329	0.0983 / 0.0346
V, L, x	2	0.1855	0.0983 / 0.0872

using FPM1. The first models used the K-values taken directly from Aspen Plus, which had erroneous values when the M-W mixture only exist in a single phase, that is, K-value is not zero even though there is only one phase (either x or y is zero). The second model used the corrected K-values. The third model does not use any K-values because Model 2, which is the component balance, does not need the K-values.

The predictions of y from the second and third models are satisfactory, but the prediction from the first model is not. This is expected because some of the K-values being used to calculate y have errors, which would worsen and in certain cases amplify the errors of the intermediate variables. The second model, which used accurate K-values, has the smallest RMS error (0.0346) of all the three models. This is due to the fact that better K-values would enable y to be calculated accurately based on the value of x predicted by RBFN 1.

5.5.2 Hybrid RBFN-FPM-RBFN

There are four kinds of models under Type 2 hybrid models. Configurations of the models are illustrated in Figures 4.9 to 4.11.

The results for the four models, given in Table 5.11, are all satisfactory; all RMS errors for the prediction of y (ANN 2) are less than the base case. All models have

RBFN as ANN 2. The slave input variables of this model are y calculated from either FPM 1 or FPM 2 or both. On the whole, the additional network added after the FPM in all the models successfully functioned as a filter for picking out and correcting the value of y calculated by the FPM.

Comparing the two hybrid models, the addition of an RBFN after the FPM in Type 2 models is generally beneficial. For example, comparing Model 1 of Types 1 and 2, the RMS errors for the Type 2 models were lower. An exception is Model 2 of Type 1 (RMS error = 0.0872) and Model 4 of Type 2 (RMS error = 0.0874), which is essentially the same model, except for the additional RBFN. Among the models tested, all models with the correct K-values used in the FPM had very good predictions.

5.6 COMPARISON OF THE PREDICTIONS OF Y

Table 5.12 lists the RMS error of the best of the different models used to predict y. All RMS errors shown in the table represent the prediction of y in the original range that covers both the single and two-phase regions, except for the second in the list, which is

Table 5.11: Results of hybrid structure.

Intermediate Output	Model #	Total RMS	RMS (ANN 1/ANN 2)
V, L, x	1 (error in K-values)	0.1988	0.0983 / 0.1005
V, L, x	1 (correct K-values)	0.1185	0.0983 / 0.0202
V, x	2 (error in K-values)	0.1595	0.0620 / 0.0975
V, x	2 (correct K-values)	0.0855	0.0620 / 0.0235
V, x	3 (correct K-values)	0.0847	0.0620 / 0.0227
V, L, x	3 (error in K-values)	0.1807	0.0983/ 0.0824
V, L, x	4	0.1857	0.0983/ 0.0874

Table 5.12: RMS error for prediction of y using the different models.

Model	RMS error
Standard RBFN (single-phase and two-phase regions)	0.1119
Standard RBFN (two phase region only)	0.0065
Linear-Nonlinear HSNN (F)	0.0885
Linear-Nonlinear HSNN (error in K-values)	0.0857
Linear-Nonlinear HSNN (correct K-values)	0.0344
Nonlinear-nonlinear HSNN	0.0820
Serial RBFN-RBFN	0.1031
Serial RBFN - Output-tuned HSNN (V)	0.0114
Serial RBFN - Output-tuned HSNN (Component Balance)	0.0074
Hybrid RBFN - FPM (correct K-values)	0.0346
Hybrid RBFN-FPM-RBFN (correct K-values)	0.0202

the prediction of y in the two-phase envelope. The results shown in Table 5.12 are divided into four groups:

- standard RBFN
- completely driven HSNN
- serial networks
- hybrid networks

The result of the base case, which is the prediction of y over the original region using standard RBFN, had the highest RMS error (0.1119) among the groups of models. The RBFN model for the two-phase region, on the other hand, had the lowest RMS error (0.0065) among all models. Figures 5.2 and 5.3 illustrate the plots of actual y versus predicted y for both models. In the plots, the diagonal line is where the test data points should be when the predicted y is the same as the actual or desired y. As seen in

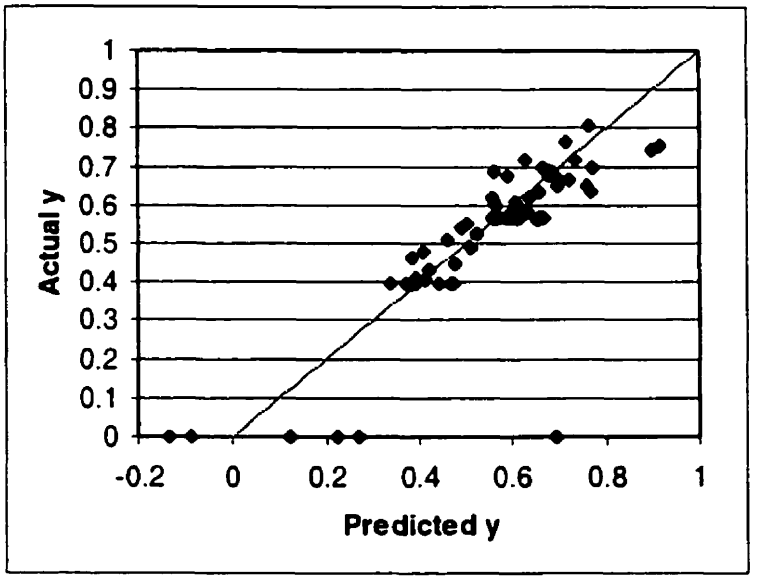

Figure 5.2: Plot of y predicted using standard RBFN (base case) (RMS = 0.1119).

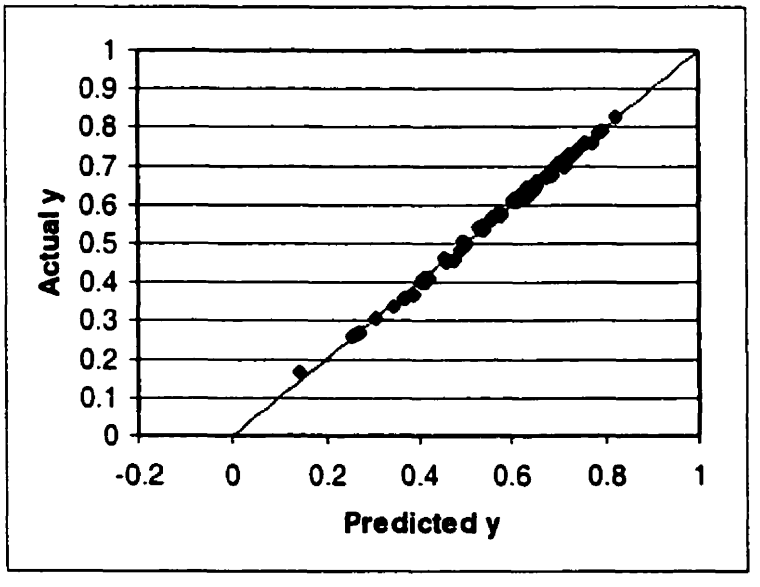

Figure 5.3: Plot of y in the two-phase envelope only (RMS = 0.0065).

Figure 5.2 for the base case, the six points in the test set where the actual y is zero are far off, with the largest difference at about 0.7. The rest of the points, which are in the two-phase region, are scattered near the diagonal. As seen in Figure 5.3 for the model for the two-phase region, almost all the test points are on the diagonal. This shows that the addition of points in the single-phase region "confused" the RBFN during training. Consequently, the network failed to perform even in predicting the points in the two-phase region.

All the completely driven HSNN models yielded better results than the base case. Four models were tested in this group:

1. **Linear-nonlinear HSNN with F as the slave input.** This model had a more than 20% reduction in RMS error. Referring to Figure 5.4, the six points where the actual y is zero were all predicted closer to zero than the standard RBF model. The largest difference is about 0.66. The points in the two-phase region are also closer to the diagonal than the base case.
2. **Linear-nonlinear HSNN, with K-values that had errors in the single-phase regions, as the slave input.** This model had similar results to the first model, as seen in Figure 5.5. The prediction of this model for test data in the two-phase region, however, is slightly off-diagonal, with the predicted values being slightly higher than the actual y. However, this is not a trend because different learning rates and initial values of the weights and biases yield slightly different predictions. For example, Figure 5.6 illustrates the results obtained using the same learning rate as the prediction in Figure 5.5, but would have different initial values of weights and biases because these values were randomly generated by an initialisation algorithm.
3. **Linear-nonlinear HSNN, with the correct values for K used as the slave input.** This model yielded the lowest RMS error (0.034) among all the models in this group. As seen in Figure 5.7, the six test data in the single-phase region were predicted close to zero, which effectively lowered the RMS error.

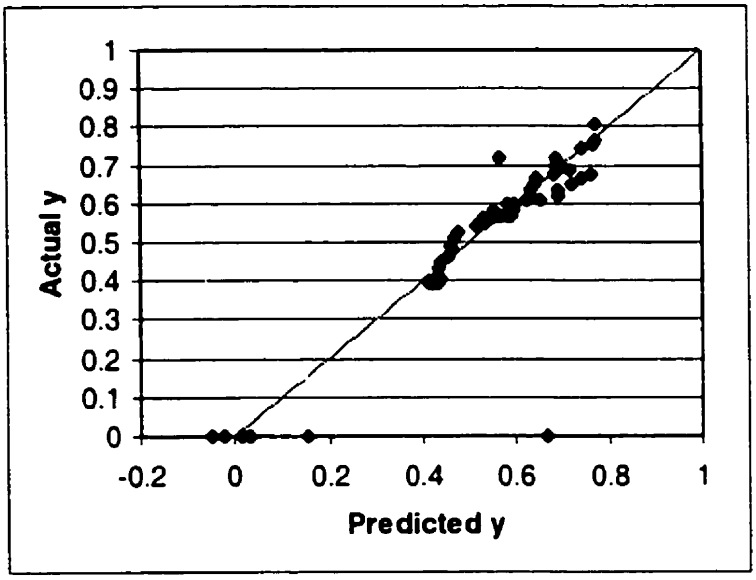

Figure 5.4: Plot of y predicted using linear-nonlinear HSNN (F slave) (RMS = 0.0885).

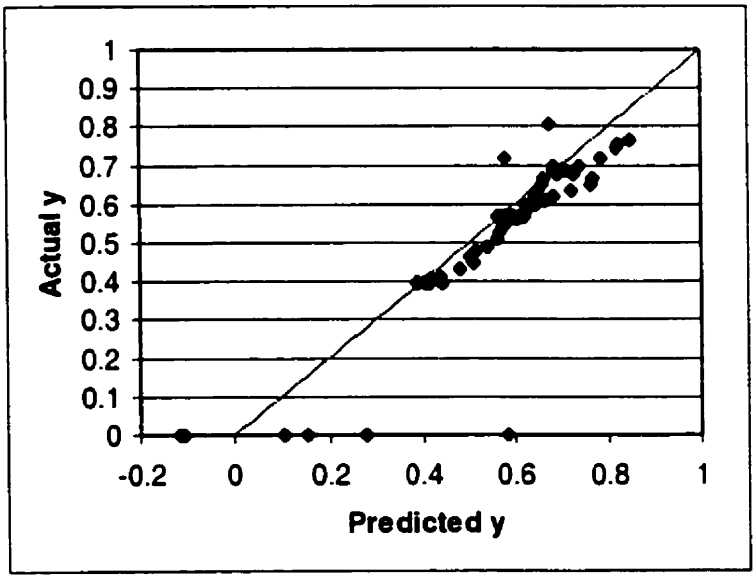

Figure 5.5: Plot of y predicted using linear-nonlinear HSNN (K-values with errors as slave input) (RMS = 0.0857).

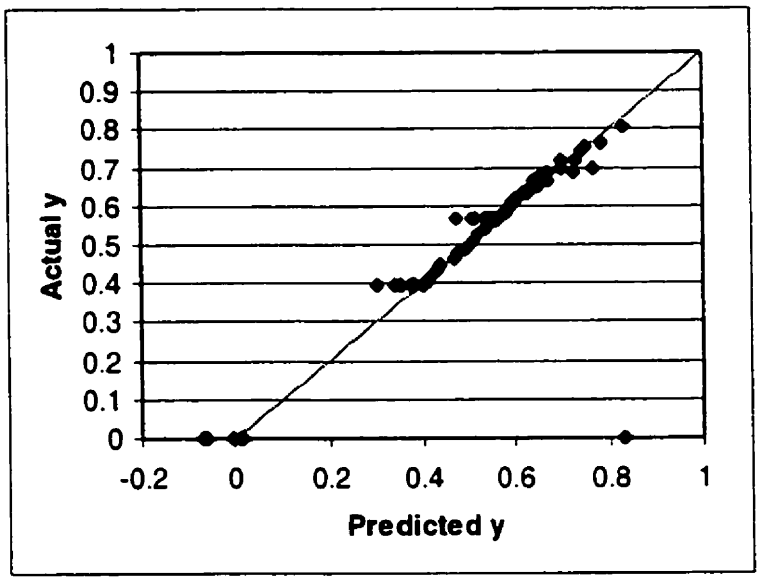

Figure 5.6: Plot of y predicted using linear-nonlinear HSNN (K-values with errors as slave input) (RMS = 0.0864).

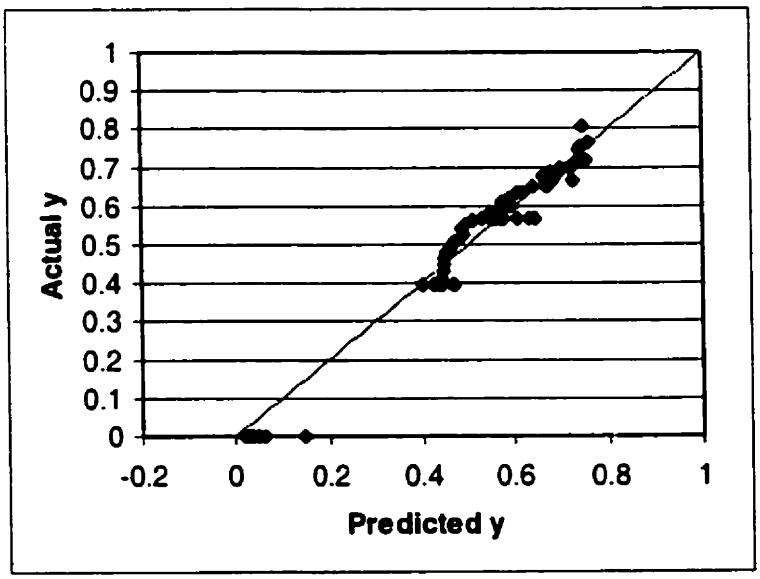

Figure 5.7: Plot of y predicted using linear-nonlinear HSNN (K-values without errors as slave input) (RMS = 0.0344).

4. **Nonlinear-nonlinear HSNN, with z as the slave input.** From Figure 5.8, the prediction of this model is better than the first and second model. In fact, in the two-phase region, the results seemed slightly better than or at least comparable to the third model. Unfortunately, the model could not give good predictions for the test data in the single-phase region, resulting in a RMS error of 0.0820.

The predictions obtained from the completely driven HSNN models were more satisfactory than the base case. Unlike standard RBFN, the hierarchical structure of the models were able to eliminate the "confusion" of predicting y in the two-phase region even in the presence of data in the single phase region. Nevertheless, only the third model, which had correct K-values used as the slave input, were able to satisfactorily

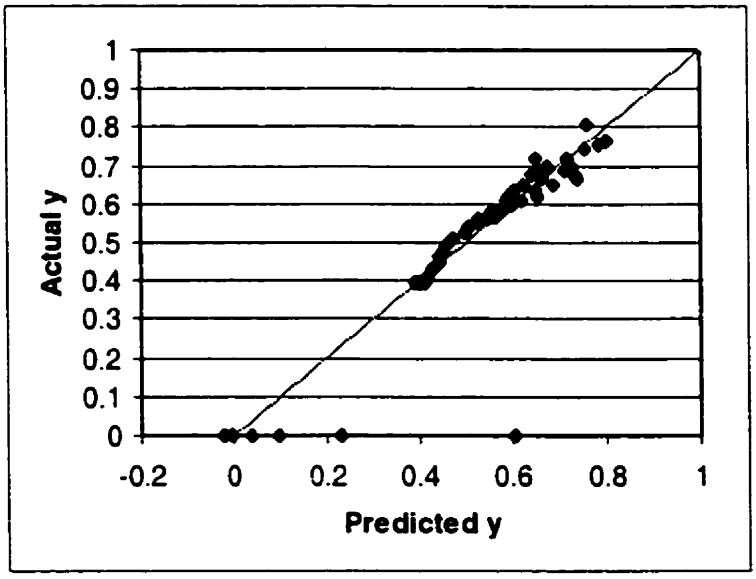

Figure 5.8. Plot of y predicted using nonlinear-nonlinear HSNN (z slave) (RMS=0.0820).

predict the test data in the single-phase region. The failure of the three other models to predict reasonably well in the single-phase region are most probably because of the absence of a more suitable slave input variable. Otherwise, in the two-phase region, all models were able to predict very well.

The predictions of the serial models are also better than the base case. Two types of serial models were tested:

1. **Serial RBFN-RBFN.** The best model prediction are plotted it Figure 5.9. The predictions are satisfactory in the two-phase region, but unsatisfactory in the single-phase region. The largest error, about 0.7, is from the prediction in the single-phase region.
2. **Serial RBFN - RBFN - output-tuned HSNN.** Both models developed under this structure had very good predictions. The lowest RMS errors for the prediction of y were 0.0114 and 0.0074 when the master inputs were V and the component balance respectively. Figure 5.10, which shows the plot when the component balance was the master input, clearly illustrates that all the test data were either on or very close to the diagonal line. The same can be said about the data in the single-phase region. The one point in the single-phase region where the predicted y is slightly greater than zero is most probably because of slight inaccuracies in the predicted values of the variables used in the constraint that was fed to the master network. Otherwise, the master network of the output-tuned HSNN managed to effectively enforce the constraint given by the master input. The results found using this model structure was the best among all other models developed.

The predictions from the serial RBFN-RBFN - output-tuned HSNN were surprisingly very good, even when a simple constraint of [V = 0 at y=0] was given. The master network was actually trained with a suitable constraint to provide the corresponding output that could effectively enforce the constraint on the output of the slave network. As seen from the results, the master network was also effective in

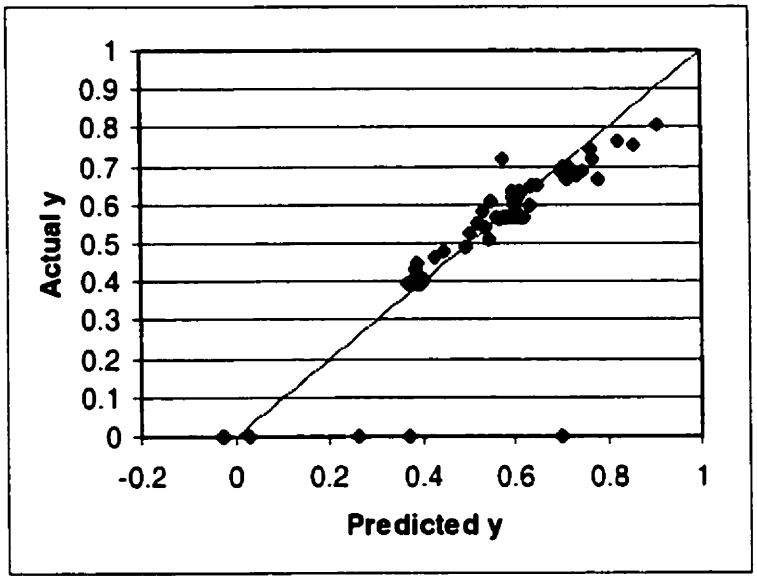

Figure 5.9: Plot of y predicted using serial RBFN-RBFN (RMS = 0.1031).

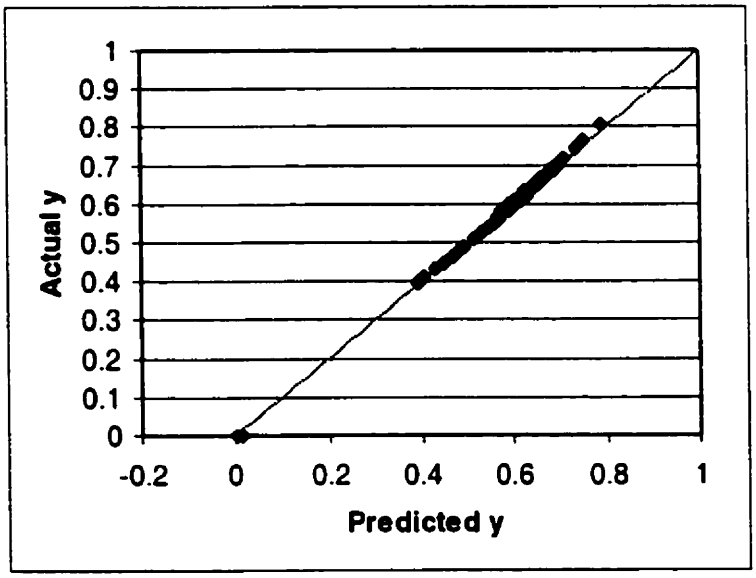

Figure 5.10: Plot of y predicted using serial RBFN-RBFN - output-tuned HSNN (FPM) (RMS = 0.0074).

zeroing the output. This is because the master network is trained so that when the input is zero, the master output and thus the gain for the slave output is zero, and thus effectively zeroing the final output. The output-tuned HSNN was not implemented on its own because of the unavailability of a suitable constraint.

The predictions of the hybrid models are also better than the base case. Two hybrid model structures were tested:

1. **Hybrid RBFN-FPM.** The best result obtained is from a model that used the equilibrium relations with the correct K-value to predict y. This is expected since having a correct K value would enable the correct calculation of y if the value of x predicted by RBFN is correct. Figure 5.11 shows that except for one point, all other test data are either on or very close to the diagonal. The point that is far from the diagonal is most probably caused by an error in the value of x (an intermediate variable) which would be amplified by K when y is calculated using the equilibrium relations.
2. **Hybrid RBFN-FPM-RBFN.** Similar to the previous model, the best result obtained is from a model that also used the equilibrium relations as the FPM with the correct K-value. The model was able to predict y well, as shown in Figure 5.12.

Overall, the serial RBFN-RBFN-output-tuned structure gave the best predictions of y. The addition of the output-tuned HSNN reduced the error in the prediction of y from 0.1031 (from the serial RBFN-RBFN) to about 0.01. The structure uses simple constraints that are easily available common knowledge for chemical processes. No complex parameter or thermodynamic estimation is needed. In addition, the training time for the output-tuned HSNN is much shorter than completely driven HSNN.

In general, almost all the models developed managed to improve the predictions of the standard RBFN models, especially for those in the two-phase region. A good prediction in the two-phase region is obtained when the RMS errors of the models in predicting y are between 0.08 and 0.09. The only way to reduce the RMS errors is by

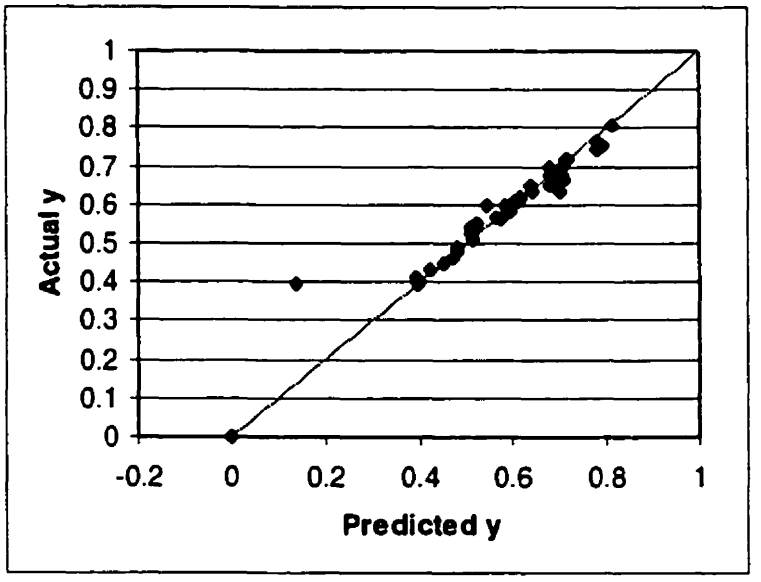

Figure 5.11: Plot of y predicted using hybrid RBFN-FPM (RMS = 0.0346).

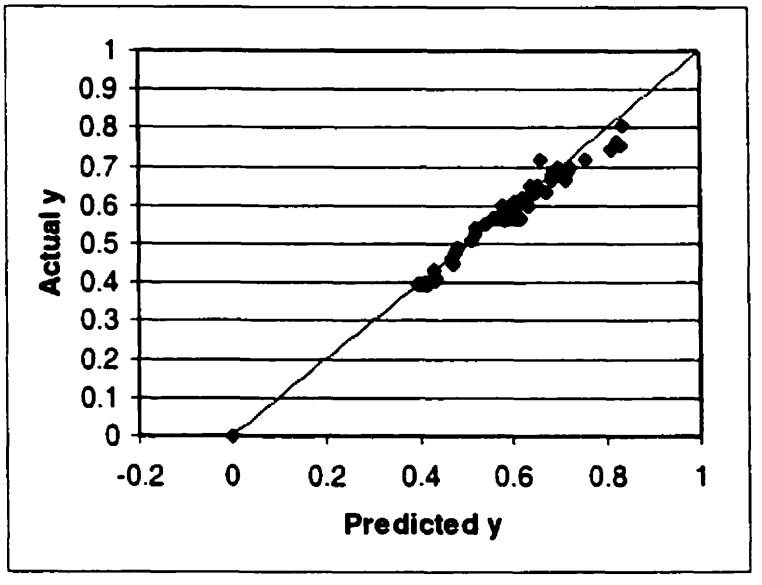

Figure 5.12: Plot of y predicted using hybrid RBF-FPM-RBF (K-values without errors as FPM input) (RMS = 0.0202).

improving the prediction of the six test data in the single-phase region, which many of the methods are unable to do so unless additional information is given. The additional information provided by the correct K-values managed to provide the information needed to properly predict the test points in the single-phase region.

5.7 CRUDE DISTILLATION TOWER

5.7.1 Sections of the Crude Tower

Changes within the operating range for a section in the crude distillation tower affect only the sections that are immediately above and below the section. This allows the crude tower model to be divided into sections where the variables that are related are grouped together, and thus make the model more manageable. In addition, as shown in modelling the methanol-water flash system, grouping suitable output variables together can yield better predictions.

The model for the crude tower is divided into the following sections:

1. top (T),
2. heavy naphtha stripper (HN),
3. kerosene stripper (K),
4. diesel stripper (D),
5. AGO stripper (A), and
6. bottom (B).

Table 3.2 lists the input and output variables of the network models for each module of the crude distillation column.

5.7.2 Comparison between RBFN and MLP

To confirm that RBFN would give a better model than standard MLP, a comparison on the RMS error and training times is made between RBFN models and MLP using backpropagation with variable μ. Table 5.13 shows the results of the two different networks using two different groups of training data for the top section of the crude tower. The average RMS error shown in Table 5.13 is the average RMS error of the five output variables for the top section. From the results, it can be seen that RBFN is superior both in prediction of the test data and training times. The rest of the sections will therefore use RBFN.

5.7.3 Grouping of Variables

To determine if the grouping of output variables had a strong influence on the prediction, the variables in the first two sections at the top of the column were predicted individually and in different groups. The results are shown in Table 5.14.

Table 5.13: Overall results for the top section of the main crude distillation column using RBFN and feedforward network with BP.

	Average RMS Error	CPU Time (sec)
RBFN w/ 300 training data	0.0030	22.85
RBFN w/ 150 training data	0.0063	9.83
BP w/ 300 training data	0.0338	1397
BP w/ 150 training data	0.0383	691

Table 5.14: RMS errors of variables of top and HN sections of the crude tower.

Outputs	Overall RMS Error	Individual RMS Error
Ttop	0.0048	
PAT	0.0140	
RR	0.0046	
Ovhd	0.0029	
Qcond	0.0033	
TtopH	0.0039	
TbotH	0.0039	
PAH	0.0099	
IBPH	0.0046	
FBPH	0.0046	
RhoH	0.0076	
IBPH, RR, Qcond	**0.0134**	**0.0051, 0.0042, 0.0041**
Ttop,RR,Qcond	**0.0067**	**0.0023, 0.0035, 0.0009**
Ttop, Ovhd, RR, Qcond, PAT	0.0146	0.0014, 0.0015, 0.0025, 0.0017,0.0075
TtopH, TbotH, PAH, IBPH, FBPH, RhoH	0.0292	0.0021,0.0028,0.0121,0.0029,0.0019,0.0074

From the table, it can be seen that for almost all the variables, the RMS errors are smaller when the variables are grouped together in a suitable combination. For example, the RMS errors for variables at the top of the column, Ttop, Ovhd, RR, Qcond and PAT are 0.0048, 0.0029, 0.0046, 0.0033 and 0.0140 respectively when predicted individually, compared to 0.0014, 0.0015, 0.0025, 0.0017 and 0.0075 respectively when predicted together. The same is also true with the variables in the HN section.

The results also show that it is not advisable to combine unrelated variables. For example, comparing the two variable combinations that are highlighted in the table, the combination with IBPH, which is in a different section than RR and Qcond, the RMS error for RR and Qcond are higher than when the variables were combined with Ttop.

5.7.4 Overall Prediction

The RMS errors for all output variables of the crude tower are given in Table 5.15. Output variables in the same section are grouped and predicted together. The results, as seen in the table, are very good. All the RMS errors are in the order of 10^{-3}, and some are even smaller. This shows that RBFN is suitable for predicting the output variables of the crude tower, and there is no need for more complex models.

Table 5.15: Overall result for all sections in the crude distillation tower.

Outputs (y1,y2,y3,y4,y5,y6)	Total RMS Error	Individual RMS Error
Top column section Ttop, Ovhd, RR, Qcond, PAT	0.0146	0.0014,0.0015,0.0025,0.0017,0.0075
HN stripper section TtopH, TbotH, PAH, IBPH, FBPH, RhoH	0.0292	0.0021,0.0028,0.0121,0.0029,0.0019 0.0074
Kerosene stripper section TtopK, TbotK, FPKero, IBPK, FBPK	0.0174	0.0018,0.0017,0.0021,0.0021, 0.0097
Diesel stripper section TtopD, TbotD, IBPD, FBPD, PourD, PAD	0.0210	0.0037,0.0036,0.0052,0.0054,0.0030 0.0001
AGO stripper section TtopA, TbotA, IBPA, FBPA, PourA, PAA	0.0133	0.0005,0.0007,0.0021,0.0050,0.0012 0.0038
LSWR section (Bottom of main column) TBOT, PourL	0.0098	0.0038, 0.0060

Referring to Table 5.15, the results are surprisingly much better than those obtained for the M-W flash in the original range, even though the crude tower is physically more complex than a single flash system. They are, however, comparable to the results obtained for the M-W flash in the two-phase region. Although the crude tower has more variables and more components involved, the model is continuous within the operating range. Note also that the components of crude oil are thermodynamically closely related. These are most possibly the major contributing factors that enabled the excellent prediction of the output variables of the crude tower.

5.7.5 Simple Range and Dimensional Extrapolation

Range extrapolation takes place when one of the input variables to a model is applied outside the range that it was trained for. Dimensional extrapolation takes place when a variable that was not part of the input variable during identification (because it was constant) varies during the use of the model [te Braake et al. 1998]. To ensure that the RBFN model can perform satisfactorily in both range and dimensional extrapolation, the kerosene section of the column was tested.

A model for the kerosene section was developed with five input variables instead of six, leaving out the kerosene stripping steam in the input to test for dimensional extrapolation. The stripping steam rate was fixed at the normal operating point. A test data set was developed with the stripping steam at the maximum and minimum operating range.

To test for range extrapolation, a test data set was developed with the stripping steam at 10% above the maximum and 10% below the minimum steam rate. Although this condition is avoided in practice, getting a model that would be feasible just outside its range is important because certain optimisation algorithms cross over constraints slightly in an effort to reach the optimum value.

Table 5.16 shows the results for both range and dimensional extrapolation. The results obtained for range extrapolation was close to the original results. There is less than 15% increase in the total RMS error. This is good because this shows that the RBFN model is able to provide a reasonable prediction should the optimiser crosses over a constraint slightly. There was, however, a nearly three fold increase in total RMS error for the dimensional extrapolation case. Nevertheless, the predictions are still satisfactory and can be accepted for use because all the individual RMS errors are around 1% or less. Therefore, the RBFN model is suitable modelling the crude tower for RTO.

5.7.6 Objective Function for RTO

This section gives an example of a possible economic objective function for RTO for the crude oil distillation column. A profit based objective function consists of the total product values after deducting the costs of feed and utilities, as shown in Equation 6.1.

Table 5.16: RMS errors for range and dimensional extrapolation.

Outputs (y1,y2,y3,y4,y5)	Total RMS Error	Individual RMS Error
Original result TtopK, TbotK, FPKero, IBPK, FBPK	0.0174	0.0018,0.0017,0.0021,0.0021, 0.0097
Dimensional extrapolation TtopK, TbotK, FPKero, IBPK, FBPK	0.0515	0.0110,0.0242,0.0024,0.0034, 0.0105,
Range extrapolation TtopK, TbotK, FPKero, IBPK, FBPK	0.0200	0.0028,0.0026,0.0027,0.0029, 0.0090

$$\max_{x}\left[\sum_{i=1}^{6} P_i - C_{feed} - C_{utilities}\right] \qquad (6.1)$$

subject to $g(x) \geq 0$

$h(x) = 0$

where P_i = product stream values;

C_{feed} = feed stream cost;

$C_{utilities}$ = total utility costs;

$g(x)$ = set of process inequality constraints;

$h(x)$ = set of equality constraints represented by the process models.

x = the decision variables, which are the draw-off flow rates.

The product stream values are simply the product draw-off flowrates, F_i, multiplied by the respective prices, D_i, as shown in Equation 6.2:

$$P_i = (D_i)\,(F_i) \qquad \text{for } i = 1,\ldots,6 \qquad (6.2)$$

The feed costs are the flow rates of the condensate and crude oil streams, F_j, multiplied by the respective prices, D_j, as shown in Equation 6.3:

$$C_{feed} = (D_j)\,(F_j) \qquad \text{for } j = 1, 2. \qquad (6.3)$$

The utility costs are consists of the reboiler (in the HN side-stripper) and condenser duties (top of column) and the stripping steam rates (main column, kerosene stripper, diesel stripper and AGO stripper), multiplied by the respective prices, D_k, as shown in Equation 6.4:

$$C_{utilities} = (D_k)\,(F_k) \qquad \text{for } k = 1, 2. \qquad (6.4)$$

The inequality constraints, g(x), may consist of the product quality constraints and equipment constraints. Among the product quality constraints, for example, is a certain range of flash point that is specified for kerosene and a certain specific gravity that is specified for heavy naphtha. The equipment constraints would include the maximum and minimum feed and product flow rates because of the constraints from the pumps that are used. Similar to the equality constraints, h(x), the values of g(x) are all calculated from the process models.

5.8 SUMMARY

For the flash systems, RBFNs were able to satisfactorily predict all output variables, except y. The difficulty in predicting y was mainly due to the discontinuity between the two-phase region and the single-phase regions. The excellent results obtained from the prediction of the output variables restricted within the two-phase region, shows that the RBFN can sufficiently model nonlinear systems when there is no discontinuity.

The RBFNs were able to predict all the output variables for the crude distillation tower very well. The RMS errors obtained were, in fact, equivalent in order of magnitude to those obtained in the M-W flash prediction restricted to the two-phase region. This can be explained by the fact that all training and testing data for the crude tower are within the operating region. In addition, the mixtures in crude oil are thermodynamically ideal.

The difference in formulating the RBFN models between the M-W system and the crude tower is mainly in the high number of output variables and the multiple sections of the crude distillation column. Nevertheless, once divided into sections, better results were obtained when the output variables from the same section of the tower were predicted together, which is a similar phenomenon with the M-W system.

The difficulty in predicting y from the M-W flash system showed that certain variables need more complex models. If additional accurate thermodynamic information, like the K-values, are available, then using a hybrid RBFN-FPM model is one of the simplest methods. HSNN is also a suitable method, especially output-tuned HSNN, which uses simple, readily available information, and does not need additional thermodynamic information.

The following summarises the basic guidelines that should be followed to formulate a connectionist model for a chemical process:

1. **Select a suitable standard ANN model.** A common, standard starting point is the MLP. However, the systems studied here were better modelled with RBFN. Therefore, we recommend that RBFN also be tested.
2. **Grouping of variables.** If there is a large number of output variables, not all of them can be predicted with one ANN model because the model would be too large and become inaccurate. An efficient way is to group related variables together since this would most probably yield better predictions while at the same time avoid "the curse of dimensionality".
3. **Serial predictions.** If there are variables that are easier to predict and can aid in the prediction of the other output variables, it is more efficient to predict them first, and then use the predicted values as part of the inputs. This type of serial prediction has been shown to yield better results with the aid of the additional information from previously predicted variables.
4. **Difficult to predict variables with extensive prior information.** If there is a difficult to predict variable and extensive information is available, such as thermodynamic or mass transfer parameters, then a hybrid ANN-FPM model can be used. If more than one FPM is available, a hybrid ANN-FPM-ANN can also be investigated. However, models developed for RTO should not, as far as possible, need rigorous numerical solutions. These models should try to keep to the simple,

straightforward calculations of an ANN model. If this is not possible, then use one of the models recommended in step 5.

5. **Difficult to predict variables, especially those that are discontinuous and without extensive prior information.** For this type of variable, HSNN is recommended. There are two options available:
   - With simple constraints. If there are simple constraints, for example, in the form of mass, component, or energy balance, then output-tuned HSNN is recommended. Simple zeroing constraints can also be used.
   - Without constraints. An input variable, which strongly influences the output variable, is used as the slave input of a completely driven HSNN. If the relationship is linear, then the linear-nonlinear HSNN is recommended. If the relationship is non-linear, then the nonlinear-nonlinear HSNN is recommended.

CHAPTER 6

CONCLUSIONS

6.1 CONCLUSIONS

The main objective of this research has been on developing different types of connectionist models that are appropriate for RTO applications. These models, which are multivariable in nature because they represent the complete process, should have a dominant ANN structure to fulfil the desired characteristics of the models for RTO. This research also investigated the use of models that have the capabilities of imbedding easily available, prior information into the structure of the model, like HSNN, for instance.

In accordance with the objective, the findings of this study can be summarised as follows:

- RBFN is suitable for modelling nonlinear chemical processes. The RBFN was able to model the flash systems and the crude distillation tower well, with the exception of output variables with discontinuity. The RBFN model for the crude tower was also able to satisfactorily perform dimensional and range extrapolation, which are important in RTO applications.
- To develop models for RTO, output variables that are related should be grouped together as this would most probably lead to better predictions. Decomposing large multivariable systems into smaller modules is also necessary so that the developed

models are manageable. In addition, grouping unrelated variables together causes unwanted degeneration of the model, and as such is not advisable.

- The HSNN structures are efficient in embedding readily available knowledge, especially in predicting variables with discontinuities and constraints. The completely driven HSNN is suitable when there is a variable with a direct and strong influence on the behaviour of the output variable. The output-tuned HSNN is suitable for consolidating constraints into the network model.
- Serial network models can improve predictions; however, when only standard networks are used, these models cannot handle discontinuities.
- Hybrid ANN-FPM models are suitable for RTO when there are easily solved FPM with the availability of accurate parameters used in the FPM. These information, however, are more difficult to obtain.

Therefore, ANN and grey box ANN models have the capabilities to model chemical processes for RTO. As shown in this study, various methods exist to overcome difficulties in modelling certain variables. Readily known information can be incorporated in different ways. Embedding prior information into the network structure provides an efficient means to come up with a better model, as in the case of HSNN. Most importantly, all the models developed here can be easily updated and maintained, which makes them suitable for the process industry.

6.2 CONTRIBUTIONS

In summary, we have accomplished the following through this research work:

1. Illustrated techniques to incorporate available FPM information into ANN models.
2. Developed a general network structure that can be used to incorporate constraints and discontinuities.

3. Developed an ANN model of an industrial crude tower.
4. Showed that the physical complexity of a process does not always correlate with the ANN model complexity.
5. Opened a window for the use of more complex network structures in chemical process modelling.

6.3 RECOMMENDATIONS

Several aspects of this research work can be enhanced or developed even further. Possible future extensions include:

1. Explore other methods to incorporate existing information into the structure of the network models. There are numerous methods in the study of neural networks that have the potential to be beneficial and have not yet been applied to chemical processes. In addition, for processes with abrupt changes, the possibility of training a classification network to determine the region, L-only, V-L or V-only, prior to calculation of the composition and quantity in each phase.
2. The models developed here could be applied to other complex chemical processes, especially those with discontinuities. An example is the polymerisation of styrene [Yang et al. 1999].
3. Several on-line issues may also be studied, including:
   - simulate on-line implementation and study possible actual implementation of the models to real world application such as the Petronas Refinery in Malaysia.
   - investigate on-line versus off-line updating.

REFERENCES

Abilov, A., and Zeybek, Z., "Use of Neural Network for Modeling of Non-linear Proccess Integration Technology in Chemical Engineering", Chemical Engineering and Processing, 39, pp. 449-458 (2000).

Altissimi, R., Brambilla, A., Deidda, A., and Semino, D., "Optimal Operation of a Separation Plant using Artificial Neural Networks", Computers in Chemical Engineering, Vol. 22, pp. S939-S942 (1998).

Aspen Technology Inc., Cambridge, Massachusetts. ASPEN PLUS User Guide Volume 1, release 9.2 edition, November 1995.

Aspen Technology Inc., Cambridge, Massachusetts. SPEEDUP User Manual Volume 1, release 5.5d edition, March 1995.

Baratti, R., Vacca, G., and Servida, A., "Neural Network Modelling of Distillation Columns", Hydrocarbon Processing, pp 35-38 (June 1995).

Baughman, D., and Liu, Y., Neural Networks in Bioprocessing and Chemical Engineering, Academic Press, San Diego, Ca., (1995).

Bhat, N., and McAvoy, T., "Determining Model Structure for Neural Models by Network Stripping", Computers and Chemical Engineering, pp. 271-282 (1992).

Billings, S., and Zheng, G., "Radial Basis Function Networks Configuration Using Genetic Algorithms", Neural Networks, 8(6), pp. 877-890 (1995).

Bittanti, S., and Savaresi, S., "Hierarchically Structured Neural Networks: A Way to Shape a 'Magma' of Neurons", Journal of the Franklin Institute, 335B(5), pp. 929-950 (1998).

Bulsari,, A., Lewandowski, J., and Palosaari, S., "System Identification of an Adsorption Process using Neural Networks", Proceedings IFAC Advanced Control of Chemical Processes, Kyoto, Japan, pp 53-57 (1994).

Busson, P., Nobrega, R., and Varela, J., "Modular Neural Networks for On-line Event Classification in High Energy Physics", Nuclear Instruments and Methods in Physics Research A, 410, pp. 273-283 (1998).

Calderon, Z., Espuna, A., and Puigjaner, L., "Waste Analysis and Minimization in Batch and Semibatch Reactor Operation through Dynamic Simulation and Neural Networks", Computers and Chemical Engineering, 22, pp. S977-S980 (1998).

Chen, K., Yang, L., Yu, X., and Chi, H., "A Self-generating Modular Neural Network Architecture for Supervised Learning", Neurocomputing, 16, 33-48 (1997).

Chen, S., Cowan, C., and Grant, P., "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks", IEEE Transactions on Neural Networks, 2(2), pp. 302-309 (1991).

Chen, T., and Chen, H., "Approximation Capability to Functions of Several Variables, Nonlinear Functionals, and Operatiors by Radial Basis Function Neural Networks", IEEE Transactions on Neural Networks, 6(4), pp.904-910 (1995).

Cheung, T., Kwapong, O., and Elsey, J., "Building Empirical Models of Process Plant Data By Regression or Neural Network", Proceedings American Control Conference, Chicago, pp.1922-1925 (1992).

Cristina, H., Aguiar, I., and Filho, R., "Modeling and Optimization of Pulp and Paper Processes using Neural Networks", Computers and Chemical Engineering, 22, pp. S981-S984 (1998).

Cubillos, F., Alvarez, P., Pinto, J. C. and Lima, E., "Hybrid-neural Modelling for Particulate Solid Drying Processes", Powder Technology 87, pp 153-160 (1996).

Cutler, C., and Perry, R., Computers and Chemical Engineering, 7, pp. 663-667 (1983).

Elkamel, A., Al-Ajmi, A., and Fahim, M., "Modeling the Hydrocracker Process using Artificial Neural Networks", Petroleum Science and Technology, 17(9&10), 931-954 (1999).

Elkamel, A., "An Artificial Neural Network for Predicting and Optimizing Immiscible Flood Performance in Heterogeneous Reservoirs", Computers Chemical Engineering, 22(11), pp. 1699-1709 (1998).

Elkamel, A., Karkoub, M., and Gharbi, R., "A Neural Network Prediction Model of Fluid Displacements in Porous Media", Computers in Chemical Engineering, 20, Supplement, pp S515-S520 (1996).

Freeman, J., and Saad, D., "On-line Learning in Radial Basis Function Networks", Neural Computation, 9(7), pp. 1601-1622 (1997).

Fu, P., and Barford, J., "A Hybrid Neural Network - First Principles Approach for Modelling of Cell Metabolism", Computers and Chemical Engineering, 20(6/7), pp. 951-958 (1996).

Gallineri, P., "Training of Modular Neural Net Systems", Handbook of Brain Theory and Neural Networks, Arbib, M., ed., MIT Press, Cambridge, MA., pp. 582-585 (1995).

Gontarski, C., Rodrigues, P., Mori, M., and Prenem, L., "Simulation of an Industrial Wastewater Treatment Plant using Artificial Neural Networks", Computers and Chemical Engineering, 24, pp. 1719-1723 (2000).

Gupta, S., Liu, P., Svoronos, S., Sharma, R., Abdel-Khalek, N., Cheng, Y., and El-Shall, H., "Hybrid First-Principles/ Neural Networks Model for Column Floatation", AICHE Journal, 45(3), pp. 557-566 (1999).

Hassoun, M.H., (1995), "Fundamentals of Artificial Neural Networks", The MIT Press, Cambridge, Massachusetts.

Haykin, S., "Neural Networks A Comprehensive Foundation", Maxwell Macmillan International, New York (1994).

Haykin, S., "Neural Networks A Comprehensive Foundation", Maxwell Macmillan International, New York (1999).

Himmelblau, D., "Applications of Artificial Neural Networks in Chemical Engineering", Korean Journal of Chemical Engineering, 17(4), pp.373-392 (2000).

Hornik, K., Stinchcombe, M., and White, H., "Multilayer Feedforward Neural Networks are Universal Approximators", Neural Networks, vol. 2, pp 359-366 (1989).

Hussain, M, "Review of the Applications of Neural Networks in Chemical Process Control - Simulation and Online Implementation", Artificial Intelligence in Engineering, 13, pp. 55-68 (1999).

Jang, J., Sun, C., and Mizutani, E., Neuro-Fuzzy and Soft Computing, Prentice Hall, New Jersey (1997).

Jang, S. Babu, J., and Mikai, H., "On-line Optimization of Constrained Multivariable Chemical Processes", AICHE Journal, 33, pp. 26-35 (1987).

Jenkins, R. and Yuhas, B., "A Simplified Neural Network Solution Through Problem Decomposition: The Case of the Truck Backer-Upper", IEEE Transactions on Neural Networks, 4(4), pp. 718-720 (1993).

Jordan, M., and Jacobs, R., "Hierarchical Mixtures of Experts and the EM Algorithm", Neural Computation, 6, pp. 181-214 (1994).

Jordan, M., and Jacobs, R., "Modular and Hierarchical Learning Systems", The Handbook of Brain Theory and Neural Networks, Arbib, M., ed., MIT Press, Cambridge, MA., pp. 579-582 (1995).

Lanouette, R., Thibault, J., and Valade, J., "Process Modeling with Neural Networks using Small Experimental Datasets", Computers and Chemical Engineering, 23, pp 1167-1176 (1999).

Latrille, E., Corrieu, G., and Thibault, J., "Neural Network Models for Final Process Time Determination in Fermented Milk Production", Computers in Chemical Engineering, 18(11/12), pp 1171-1181 (1994).

Leonard, J., Kramer, M, and Ungar, L., "A Neural Network Architecture that Computes Its Own Reliability", Computers and Chemical Engineering, 16(9), pp. 819-835 (1992).

Leonard, J., and Kramer, M, "Improvement of the Backpropagation Algorithm for Training Neural Networks", Computers and Chemical Engineering, 14(3), pp. 337-341 (1990).

Lin, C.T. and C.S. Lee, "Neural Fuzzy Systems", Prentice Hall Publishing (1995).

Lu, B. and Ito, M., "Task Decomposition Based on Class Relations: A Modular Neural Network Architecture for Pattern Classification", Lecture Notes in Computer Science, 1240, 330-339 (1997).

Luo, W., and Billings, S., "Structure Selective Updating for Nonlinear Models and Radial Basis Function Neural Networks", Inter. J. Adapt. Control Signal Processing, 12, 325-345 (1998).

MacMurray, J., and Himmelblau, D., "Modeling and Control of a Packed Distillation Column Using Artificial Neural Networks", Computers in Chemical Engineering, 19(10), pp 1077-1088 (1995).

Mandlischer, M., Geyer, H., and Ulbig, P., "Neural Networks and Evolutionary Algorithms for the Prediction of Thermodynamic Properties for Chemical Engineering", Lecture Notes in Computer Science, Vol. 1585, pp. 106-113, (1999).

Mavrovouniotis, M., and Chang, S., "Hierarchical Neural Networks", Computers and Chemical Engineering, pp.347-370 (1992).

Meghlaoui, A., Thibault, J., Bui, R., Tikasz, L., and Santerre, R., "Neural Networks for the Identification of the Aluminium Electrolysis Process", Computers in Chemical Engineering, 22(10), pp. 1419-1428 (1998).

Molga, E., van Woezik, B., and Westerterp, K., "Neural Networks for Modelling of Chemical Reaction Systems with Complex Kinetics: Oxidation of 2-Octanol with Nitric Acid", Chemical Engineering and Processing, 39, pp.323-334 (2000).

Moody, J., and Darken, C., "Fast Learning in Networks of Locally-Tuned Processing Units", Neural Computation, 1, pp. 281-294 (1989).

Nascimento, C., and Giudici, R., "Neural Network Based Approach for Optimisation Applied to an Industrial Nylon-6,6 Polymerisation Process", Computers and Chemical Engineering, 22, pp S595-S600 (1998).

Nascimento, C., Giudici, R., and Scherbakoff, N., "Modeling of Industrial Nylon-6,6 Polymerization Process in a Twin-Screw Extruder Reactor. II Neural Networks and Hybrid Models", Journal of Applied Polymer Science, Vol. 72, 905-912 (1999).

Nascimento, C., Giudici, R., and Guardani, R., "Neural Network Based Approach for Optimization of Industrial Chemical Processes", Computers and Chemical Engineering, 24, pp. 2303-2314 (2000).

Naysmith, M., Ph.D. Thesis, Dept. of Chemical Engineering, University of Waterloo (1997).

Naysmith, M., and Douglas, P., "Review of Real Time Optimization in the Chemical Processing Industry", Developments in Chemical Engineering and Mineral Processing, 3(2), pp. 67-87 (1995).

Piron, E., Latrille, E., and Rene, F., "Application of Artificial Neural Networks for Crossflow Microfiltration Modelling: Black Box and Semi-physical Approaches", Computers and Chemical Engineering, 21(9), pp. 1021-1030 (1997).

Pollock, G., and Eldridge, R, "Neural Network Modeling of Structured Packing Height Equivalent of a Theoretical Plate", Ind. Eng. Chem. Res., 39(5), pp. 1520-1525 (2000).

Psichogios, D., and Ungar, L., "A Hybrid Neural Network-First Principles Approach to Process Modelling", AICHE Journal, 38(10), pp. 1499-1511 (1992).

Reuter, M., van Deventer, J., and van der Walt, T., "A Generalized Neural-Net Kinetic Equation", Chemical Engineering Science, 48(7), pp 1281-1297 (1993).

Sabharwal, A., Bhat, N., and Wada, T., "Integrate Empirical and Physical Modelling", Hydrocarbon Processing, pp. 105-112, October 1997.

Schubert, J., Simutis, R., Dors, M., Havlik, I., and Lubbert, A., "Bioprocess Optimization and Control: Application of Hybrid Modelling", Journal of Biotechnology, 35, pp. 51-68 (1994).

Schubert, J., Rimvydas, S., Dors, M., Havlik, I., and Lubberi, A., "Hybrid Modelling of Yeast Production Processes – Combination of a priori Knowledge on Different Levels of Sophistication", Chemical Engineering Technology, 17, pp 10-20 (1994).

Seader, J., and Henley, E., "Separation Process Principles", John Wiley & Sons Inc., New York, pp. 178-180 (1998).

Sharma, R., Singhal, D., Ghosh, R., and Dwivedi, A., "Potential Applications of Artificial Neural Networks to Thermodynamics: Vapor-Liquid Equilibrium Predictions", Computers in Chemical Engineering, 23, pp. 385-390 (1999).

Shene, C., Diez, C., and Bravo, S., "Neural Networks for the Prediction of the state of Zymomonas mobilis CP4 Batch Fermentations", Computers and Chemical Engineering, 23, pp. 1097-1108 (1999).

Shertinsky, A., and Pickard, R., "On the Efficiency of the Orthogonal Least Squares Training Method for Radial Basis Function Networks", IEEE Transactions on Neural Networks, 7(1), pp.195-200 (1996).

Specht, D., "A General Regression Network", IEEE Transactions on Neural Networks, 2(6), pp. 568-576 (1991).

Sridhar, D., Seagrave, R., and Bartlette, E., "Process Modelling using Stacked Neural Networks", AICHE Journal, 42(9), pp. 2529-2539 (1996).

Sridhar, D., Bartlett, E. and Seagrave, R., "Information Theoretic Subset Selection for Neural Network Models", Computers and Chemical Engineering, 22(4/5), pp. 613-626 (1998).

Su, H., Bhat, N., Minderman, P., and McAvoy, T., Proceedings IFAC Dynamics and Control of Chemical Reactors, Maryland, USA, pp. 327-332 (1992).

Syu, M., and Chen, B., "Back-propagation Neural Network Adaptive Control of a Continuous Wastewater Treatment Process", Industrial Engineering and Chemistry Research, 37, pp. 3625-3630 (1998).

te Braake, H., van Can, H., and Verbruggen, H., "Semi-mechanistic Modeling of Chemical Processes with Neural Networks", Engineering Applications in Artificial Intelligence, 11, pp. 507-515 (1998).

Thibault, J., Acuna, G., Perez-Correa, R., Jorquera, H., Molin, P., and Agosin, E., "A Hybrid Representation Approach for Modelling Complex Dynamic Bioprocesses", Bioprocess Engineering, 22, pp. 547-556 (2000).

Thibault, J., and Grandjean, B., "Neural networks in Process Control – A Survey", Proceedings IFAC Advanced Control of Chemical Processes, Toulouse, France, pp. 251-278 (1991).

Tholundur, A., and Ramirez, W., "Neural-Network Modeling and Optimization of Induced Foreign Protein Production", AICHE Journal, 45(8), pp. 1660-1670 (1999).

Thompson, M. and Kramer, M., "Modeling Chemical Processes using Prior Knowledge and Neural Networks", AICHE Journal, 40(8), pp. 1328-1340 (1994).

Thompson, W., Martin, G., and Bhat, N., "How Neural Network Modelling Complement Those of Physical Modelling", Proceedings NPRA Computer Conference, Nov. 11-13, (1996).

Tsen, A., Jang, S., Wong, D., and Joseph, B., "Predictive Control of Quality in Batch Polymerization using Hybrid ANN Models", AICHE Journal, 42(2), pp. 455-465 (1996).

Turner, P., Montague, G., and Morris, J., "Dynamic Neural Networks in Non-linear Applications (an Industrial Application)", Computers in Chemical Engineering, 20, Supplement, pp S937-S942 (1996).

van Can, H., Hellinga, C., Luyben, K., Heijnen, J., and Braake, H., "Strategy for Dynamic Process Modelling Based on Neural Networks in Macroscopic Balances", AICHE Journal, 42(12), pp. 3403-3418 (1996).

van der Walt, T., and van Deventer, J., "The Dynamic Modelling of Ill-defined Processing Operations using Connectionist Networks", Chemical Engineering Science, 48(11), pp. 1945-1958 (1993).

Venkatasubramanian, V., and McAvoy, T., "Editorial – Neural Network Applications in Chemical Engineering", Computers and Chemical Engineering, pp. v-vi (1992).

Wang, H., Liu, G., Harris, C., and Brown, M., Advanced Adaptive Control, Pergamon Press, UK, p 51 (1995).

Whaley, A., Bode, C., Ghosh, J., and Eldridge, R., "HETP and Pressure Drop Prediction for Structured Packing Distillation Columns using a Neural Network Model", Ind. Eng. Chem. Res., 38(4), pp. 1736-1739 (1999).

Willis, M., Di Massimo, C., Montague, G., Tham, M., and Morris, A., "Artificial Neural Networks in Process Engineering", IEE Proceedings-D, 138(3), pp. 256-266 (1991).

Wilson, J., and Zorsetto, L., "A Generalized Approach to Process State Estimation using Hybrid Artificial Neural Network/Mechanistic Models", Computers and Chemical Engineering, 21(9), pp. 951-963 (1997).

Wolpert, D., "Stacked Generalization", Neural Networks, 5, pp.241-259 (1992).

Yang, S., Chung, P., and Brooks, B., "Multi-stage Modelling of a Semi-batch Polymerisation Reactor using Artificial Neural Networks", Trans. IChemE, Vol. 77, Part A, pp. 779-783 (1999).

Zhang, J., Yang, X., Morris, A., and Kiparissides, C., "Neural Network Based Estimators for a Batch Polymerization Reactor", Proceedings IFAC Dynamics and Control of Chemical Reactors (DYCORD+'95), Copenhagen, Denmark, pp 129-214 (1995).

Zhang, J., Morris, A., Martin, E., and Kiparissides, C., "Estimation of Impurit and Fouling in Batch Polymerisation Reactors through the Application of Neural Networks", Computers and Chemical Engineering, 23, pp. 301-314 (1999).

Zhao, J., Chen, B., and Shen, J., "A Hybrid ANN-ES System for Dynamic Fault Diagnosis of Hydrocracking Processes", Computers in Chemical Engineering, 21, Supplement, pp S929-S933 (1997).

Zheng. G., and Billings, S., "Radial Basis Function Network Configuration Using Mutual Information and the Orthogonal Least Squares Algorithm", Neural Networks, 9(9), pp. 1619-1637 (1996).

Zorzetto, L., Filho, R., and Wolf-Maciel, M., "Process Modelling Development through Artificial Neural Networks and Hybrid Models", Computers and Chemical Engineering, 24, pp. 1355-1360 (2000).

APPENDIX A
BACKGROUND ON NEURAL NETWORKS

This appendix provides detailed background on neural networks that is necessary to understand this research.

A.1 NEURONS AND NEURAL NETWORKS

Figure A.1 shows a neuron, which is the simplest processing element of a neural network. A neuron or a node consists of three components [Baughman and Liu, 1995; Haykin, 1994].

1. **Inputs and outputs.** This is the synapse or connections from or to other nodes. Inputs to the node, x_i, may also come from data that have been normalized. The node manipulates the inputs to yield the output, y_j, which may then be sent to more than one node.
2. **Connection weights.** The connection weights, w_{ij}, determine the influence of the input on the output of the node. In this work, the first subscript of the weight, i, refers to the input while the second subscript, j, refers to the node. Weight factors can be inhibitory (if the value is negative) or excitatory (if the value is positive). A weight factor that is close to zero will have a negligible effect on the node.
3. **Activation function.** Summation of the weighted inputs is passed through an activation function (also called squashing function or transfer function). This

function limits the amplitude range of the output. The most commonly used functions [Baughman and Liu, 1995] are the sigmoid function,

$$f(x) = 1/(1 + e^{-x}) \qquad (A.1)$$

the hyperbolic tangent function,

$$f(x) = \tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x}) \qquad (A.2)$$

and the Gaussian function,

$$f(x) = \exp(-x^2/2) \qquad (A.3)$$

As shown in Figure A.1, there may also be threshold values associated with the node. This threshold value, T_j, lowers or increase the net input of the activation function. After the inputs have gone through the neuron, as shown in Figure 2.2, the output of the network becomes:

$$y_j = \Sigma_i w_{ij} x_i - T_j \qquad (A.4)$$

The nodes can be connected in several different topologies, the most common being feedforward and recurrent networks. The nodes are arranged in layers; the network may contain a single layer, or more than one layer, in which case it becomes a multilayer network.

Multilayer Feedforward Networks. There are three types of layers: 1. input layer, 2. hidden layer, and 3. output layer. The input layer contains the source nodes. There can be more than one hidden layer. The purpose of this hidden layer is to extract higher order information from the data. The outputs of the input layer are sent to the nodes in the first layer of the hidden layer. In a feedforward network, inputs to the neurons in a layer comes from the neurons in the preceding layer. When every node in each layer of the network is connected to every node in the adjacent layer, the network is fully connected. There are also partially connected networks.

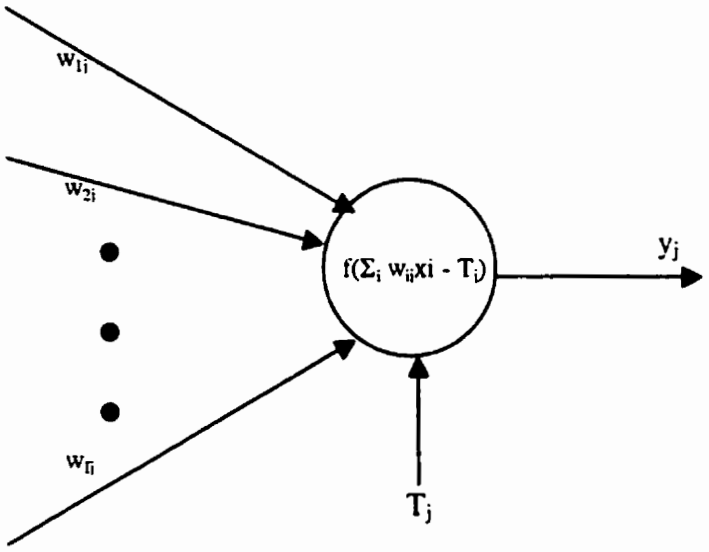

Figure A.1 Schematic diagram of a neuron.

Recurrent Networks. Recurrent network is distinct from feedforward network in that it has at least one feedback loop. Self-feedback, which is when the output of a neuron is fed back to its input, can also occur. Addition of unit delay systems or zero-order holds (denoted by z-1) in recurrent networks, is very common in dynamic modelling.

A.2 DEVELOPING ANN MODELS

There are three phases in developing ANN models [Baughman and Liu, 1995]:

1. **Training/learning phase**. Learning is the process of adjusting weight factors based on systematic and efficient trial and error. Training is the process of adjusting weight factors until the output patterns reflect the desired relationship [Baughman and Liu, 1995]. To do this, the network is repeatedly presented with a set of known input/output data. The network learns the input/output response behavior, and subsequently undergoes further training. This is the longest and most time-consuming step. It is also the most important to the success of the network.
2. **Recall phase**. The network is tested with trained data.
3. **Generalisation**. The network is tested with data that it was not trained with before. This phase will determine the interpolation/generalization capability of the network. If too many nodes are used, then there is a tendency for the network to overfit and badly generalize.

There are three main learning algorithms. Decision on which algorithm to use mainly depends on the type of problem to be solved. The three algorithms are as follows [Haykin, 1994; Lin and Liu, 1995]:

1. **Supervised learning**. It is also called active learning or learning with an external teacher. In supervised learning with every input, a corresponding desired output is given. The most common method of learning is error-correction learning, where the error, which is the difference between the network output and the desired output, is sent back to the network to correct the weights of the network to minimize the error.
2. **Reinforcement learning**. Also called learning with a critic, it does not need as much or detailed data as supervised learning. This is because feedback to the network is evaluative (critic) and not instructive (teacher). For example, the critic will give feedback evaluation like "too high", "high", or "low" to a certain output. A critic signal generator gives the external reinforcement signal.
3. **Unsupervised learning**. In this mode of learning, there is no external teacher or critic. The network relies on internal control and local information to develop its

own model without additional input information. The network "self-organize" itself by discovering patterns, features or categories in the input while adjusting its parameters. The network forms clusters by discovering similarities and differences in the object. Typically, unsupervised learning is used in classification of patterns, such as image or voice recognition.

A.3 MULTILAYER PERCEPTRONS

Multilayer perceptrons are feedforward multi-layered networks that are capable of performing just about any linear or nonlinear computation and can approximate any reasonable function arbitrarily well. A multilayer perceptron has three distinctive features [Haykin 1994]:

a) The model of each neuron in the network includes a nonlinear but smooth (ie. differentiable) activation function at the output end. The presence of this nonlinearity is important because it enables the network to map nonlinear relationships.
b) The network contains one or more hidden layers of nodes. These nodes enable the network to learn complex tasks by extracting progressively more meaningful features from the input-output patterns.
c) The network exhibits a high degree of connectivity determined by the synapses of the network. A change in the connectivity of the network requires a change in the population of synaptic connections or their weights.

Back propagation learning algorithm is one of the earliest and most common method for training multilayer perceptrons. Development of this learning algorithm was one of the main reasons for renewed interest in this area and this learning rule has become central to many current work on learning in artificial neural networks. It is used

to train nonlinear, multilayered networks to successfully solve difficult and diverse problems such as perform function approximation, pattern association and pattern classification, non-linear system modeling, time-series prediction and image compression and reconstruction [Hassoun, 1995].

Backpropagation learning consists of two passes through the different layers of the network. In the forward pass, the input pattern applied to the sensory nodes of the network propagates through the different layers. During this pass the synaptic weights of the network remain fixed. An actual output is produced as a result of the forward flow of this data. This actual response is subtracted form a desired (target) response to produce an error signal which is then propagated backward through the network against the direction of the synaptic connections. During this backward pass, the synaptic weights are all adjusted to make the actual response of the network more closer to the desired response. The error-correction scheme, therefore, works by propagating the information about the deviation from the desired output "backward" through the network, against the direction of synaptic connections.

The weights of the network can be updated by two procedures: incremental learning and batch learning. With incremental learning, the weights are updated after every presentation of an input pattern. Whereas, with batch learning, weight updating is performed only after all pattern (assuming a finite training set) have been presented. The weights of the network are updated such that the sum-squared of error of the network is minimized. This is done by continually changing the values of the network weights in the direction of steepest descent with respect to error. Steepest descent or gradient descent is one of the simplest optimization techniques and is not a very effective one. This technique may suffer from slow convergence, especially when small learning rates are used.. Alternatively, Newton's method, conjugate-direction method or quasi–newton method have been proposed to improve this algorithm [Lin and Lee, 1995].

Leonard and Kramer [1990] showed that the backpropagation algorithm, which normally uses the generalized delta rule for gradient calculations, is inefficient and has poor convergence on serial processing machines (ie. computers). Backpropagation learning is generally slow because of the characteristics of the error surface that is characterized by numerous flat and steep regions and has many troughs that are flat in the direction of search. In addition, there are local minima at error levels above the levels of the global minima of the surfaces. This causes the back propagation learning to become stuck at the local minima and converge very slowly [Lin and Lee, 1995].

To speed up the performance of backpropagation many enhancements and modifications have been proposed [Lin and Lee, 1995].

- **Weight initialization**. Owing to gradient-descent nature, backpropagation is very sensitive to initial conditions. The initial weights are typically set to small random values. The motivation to start from small weights is that, large weights tend to prematurely saturate units in a network and render them insensitive to the learning process. On the other hand randomness is introduced as a symmetry-breaking mechanism. It prevents units from adopting similar functions and becoming redundant.
- **Learning rate**. The learning rate constant, η, is essentially the step size in the direction of the gradient descent. It directly influences the convergence and effectiveness of backpropagation. No single learning constant value suitable for different training cases and $\eta$is usually chosen experimentally for each problem. If it is small, the search path will closely approximate the gradient path, but convergence will be very slow due to the large number of update steps needed to reach a local minima. On the other hand, if η is large, convergence initially will be very fast, but the algorithm will eventually oscillate and thus not reach a minimum. An efficient approach will be to use an adaptive learning rate constant, so that large steps are taken when the search is far away from a minimum with decreasing step size as the search approaches a minimum. Training time can be decreased by the

use of an adaptive learning rate which attempts to keep the learning step size as large as possible while keeping learning stable.

- **Momentum**. Momentum decreases backpropagation's sensitivity to small details in the error surface. Addition of momentum to term to the gradient-descent method suppresses oscillations occurring due to large learning rate constant. The idea is to give each weight some inertia or momentum so that it tends to change in the direction of the average downhill force that it feels. This scheme is implemented by giving a contribution from the previous step to each weight change:

$$\Delta w(t) = -\eta \nabla E(t) + \alpha \Delta w(t-1) \qquad (A.5)$$

where α is a momentum parameter and a value of 0.9 is often used.

- **Cost function**. The squared error term $(d_i - y_i)^2$ can be replaced by any other differentiable function $F(d_i, y_i)$, which is minimized when its arguments are equal, in the quadratic cost function. Based on this new cost function, a corresponding update rule can be derived. This will change the error signal for the output layer while all the other equations of the back propagation algorithm will remain unchanged.
- **Training data and generalization**. The amount of training data should be proper and sufficient. There are no rules or procedures suitable for all cases in choosing training data. One rule of thumb is that training data should cover the entire expected input space and then during the training process training-vector pairs should be randomly selected from the set.
- **Number of hidden layers and nodes**. The size of a hidden layer is usually designed experimentally. Too few neurons can lead to underfitting. Too many neurons can contribute to overfitting, in which all training points are well fit, but the fitting curve takes wild oscillations between these points. For a network of reasonable size, the size of hidden nodes need to be only a relatively small fraction of the input layer. If the network fails to converge, more hidden nodes may be

required. If it does converge, some nodes may be removed and the final size can be determined based on the overall system performance.

A.4 RADIAL BASIS FUNCTION NETWORK (RBFN)

RBFN is based on the concept of the locally tuned and overlapping receptive fields that exist in the cerebral and the visual cortex. It is also strongly rooted in the areas of interpolation and approximation theory [Jang et al. 1997, Haykin 1995]. The network is designed for nonlinear input-output mapping through training. The receptive fields of the network are radial basis functions, which can be adaptively tuned to provide sufficient overlapping for smooth mapping, but sharp enough for good approximations.

Well known for its fast, localized training, simplicity and generality, the network attracted much research, especially in the late eighties and in the nineties. Among the works include theoretical properties of RBFN [Poggio and Girosi, 1990; Chen and Chen 1995], algorithms and design [Moody and Darken, 1989; Specht, 1990; Chen et al., 1991; Billings and Zheng, 1995; Chen, et al., 1995; Zheng and Billings, 1997; Freeman and Saad, 1997; Luo and Billings, 1998], and evaluation and confidence level [Leonard, et al., 1992; Yingwei, et al., 1998]. The network performs very well for classification and multidimensional curve-fitting (approximation) problems. RBFN is also suitable for on-line applications because it can be rapidly trained. Among the applications are speech recognition, image processing, fault diagnosis, process control, time series analysis and general function approximation.

RBFN has a feedforward structure, though it differs in terms of operation from the standard feedforward neural network. Figure A.2 shows the basic structure of the network. There are three layers in the network, in which the nodes are fully connected with the nodes in the successive layer. The layers are:

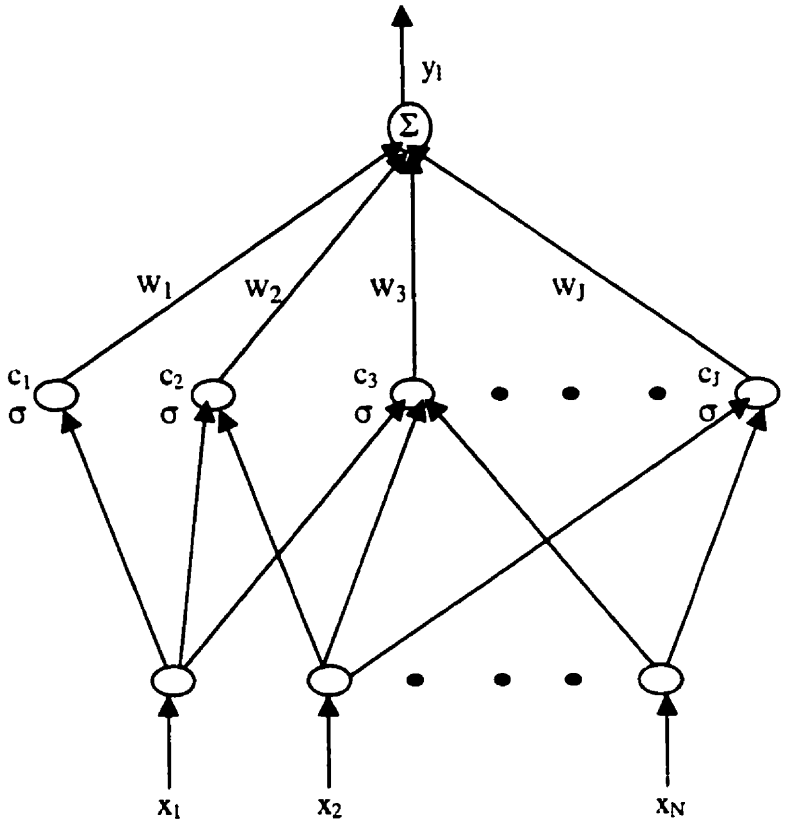

Figure A.2 General structure of a radial basis function network.

1. **First layer**. This is the input layer, which receives N inputs and sends them to the second layer. The first layer connections to the second layer are not weighted. The input to the hidden layer is therefore simply the input vector, **x**.
2. **Second layer**. This is the hidden layer, which has J nodes with Gaussian density function:

$$hj(x) = \exp[-\|x - c_j\|^2 / (2\sigma_j^2)] \qquad (A.6)$$

where c_j, $j = 1,\ldots,J$ are the RBF centers, and σ_j is the RBF width parameter. These are the receptive fields or the nodes, which make this hidden layer the most critical

layer of the RBFN. The function of this layer will be discussed further in this section.

3. **Third layer**. This is the output layer, which has linear nodes. Connections between the second and third layers are weighted. These weights can be calculated using standard backpropagation algorithm [Baughman and Liu, 1995]. The final network output is a weighted sum of the output value of the hidden layer, shown in equation A.7:

$$y_l(x) = \sum_{j=1}^{J} w_{lj} h_j(x) \qquad \text{(A.7)}$$

Figure A.3 illustrates a Gaussian receptive field. Performance of RBFN is highly dependent on the receptive field parameters, c_j and σ_j. c_j determines the location and σ_j determines the span of the activation region of the nodes in the hidden layer. Therefore, each node in the hidden layer corresponds to a unique local neighborhood in the input space. Within the region of activation, the closer the input, x_i, is to the center of the receptive field, c_j, the higher the activation level. The maximum activation level, which is one, occurs when x_i is at c_j.

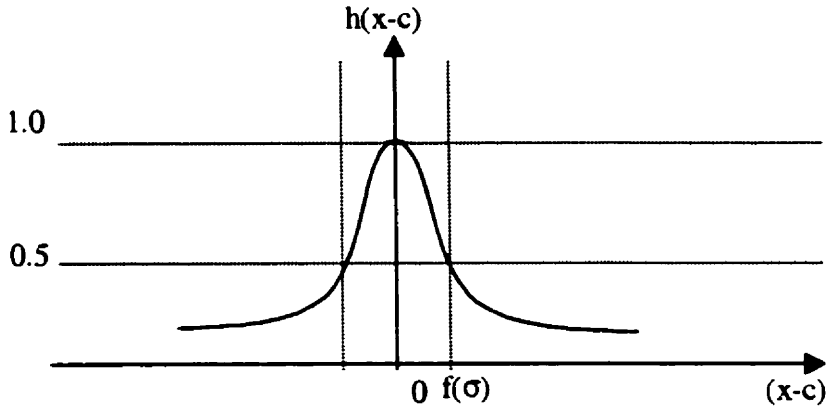

Figure A.3 Gaussian basis function

Learning for RBFN is divided into two parts. The first is the synthesis of the hidden layer, and the second is to get the weights of the output layer. The separate training scheme exploits the localized presentation of the hidden layer units, since only the nodes activated by an input need updating. The main advantage of localized training is speed. However, there is reduced generalization ability of the network unless a large number of nodes is used.

The first part of RBFN training is to get the receptive fields parameters, which are the centers, c_j, and the width of the receptive fields, σ_j. Several learning schemes exist for determining these parameters, especially for c_j. The selection of c_j is critical because the performance of the RBFN depends on the centers of the receptive field. A single value of σ is sufficient to be used for all the receptive fields. RBFN with the same σ for each receptive field in the hidden layer was theoretically proven as universal approximator [Hassoun, 1995].

The second part of RBFN training, to get the weights of the output layer, is fairly straightforward. The works surveyed used linear regression, singular value decomposition or one of the backpropagation algorithms, like the delta learning rule [Hassoun, 1995; Haykin, 1994, Leonard, et al., 1992]].

The different methods to find the receptive fields parameters can be divided into three approaches [Haykin, 1994]:

1. **Fixed c_j.** This is the simplest approach to finding c_j. Some data points from the training set are randomly chosen to be the fixed centers of the receptive field. This would be good only if the data are distributed in a representative manner for the surface approximated. In some cases, all the training data are used as centers. However, this method is not practical for problems with large amounts of data, like in speech processing. In another approach, the centers are placed on uniform course lattice along each dimension of an n-dimensional input space. However, this method is not practical for problems with high-dimensional input space. The width of the Gaussian RBF, σ, can be fixed using:

$$\sigma = d/(2J)^{1/2} \qquad (A.8)$$

where d is the maximum distance between the chosen centers, and J is the number of centers.

2. **Unsupervised selection of c_j.** This approach adaptively computes c_j. The hidden layer nodes learn to represent only parts of the input space that is densely populated by clusters of data. This results in a smaller number of nodes in the hidden layer. Moody and Darken [1989] used k-means clustering algorithm to locate J RBF centers that would minimize the sum squared error of the distance between the input training data and the center. At each time step, a random training vector, x, is selected and the center, c_j, nearest of the nearest receptive field is updated according to:

$$\Delta c_j = \rho(x - c_j) \qquad (A.9)$$

where ρ is a small positive constant.

There is no standard way of finding the number of nodes in the hidden layer, J. J is usually found by cross validation. σ is heuristically determined to get smooth interpolation [Hassoun 1995]. A very common method is the nearest neighbor heuristics, which takes the global average over all the Eucledian distances between the center of each node, i, and that of the nearest neighbor, j, as seen in the following equation:

$$\sigma = \|c_i - c_j\| \qquad (A.10)$$

A heuristic method to individually tune σ is

$$\sigma = \alpha\|c_i - c_j\| \qquad (A.11)$$

where α is a constant between 1.0 and 1.5.

3. **Supervised selection of c_j.** In this most generalized form of RBFN, all the parameters in the hidden and ouput layers are found through supervised learning. Error between the desired output and the network output is minimized, usually using a gradient descent technique. This class of method yields RBFN with good

generalization at the expense of higher training times because of the increased computation [Haykin1994].

Chen et al [1991] came up with the orthogonal least squares (OLS) algorithm, which has node-growing capability. The OLS algorithm provides a systematic method to select RBF centers. The centers are selected one at a time such that the approximation errors of the network are effectively reduced at each step. This recursive procedure is terminated once the errors have reached below a prescribed value. The MATLAB neural networks toolbox uses this algorithm to find the centers of RBF networks. An advantage of this method includes a smaller number of nodes in the hidden layer than that of RBF with randomly selected centers. Another advantage is the avoidance of numerical ill-conditioning frequently encountered in RBF with randomly selected centers.

Leonard et al. [1992] introduced the validity index network (VI net), which is an extension of RBFN. In addition to the network output, the VI net indicates when the network is extrapolating. The network is able to indicate any extrapolation based on the estimation of the local training data density.

APPENDIX B

UPDATE EQUATIONS DERIVATION FOR HSNN

B.1 Update Equations for Linear-Nonlinear HSNN

- weights of the first hidden layer of the master network, V1M,
- the biases of the first hidden nodes of the master network, C1M,
- the weights of the second hidden layer of the master network, V2M, and
- the biases of the output nodes of the master network, C2M.

The derivation of the update equation for V1M is as follows:

$$V1M_{new} = V1M - \mu \frac{\partial e}{\partial V1M}$$

$$\frac{\partial e}{\partial V1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial V1M}$$

$$e = (y - y_d)^2, \qquad \frac{\partial e}{\partial y} = 2(y - y_d) = 2e$$

$$y = b_1(U_s) + b_2$$

$$\frac{\partial y}{\partial b_1} = U_s \qquad \text{and} \qquad \frac{\partial y}{\partial b_2} = 1$$

$$b_1 = \sum_{j=1}^{6}\left[(V2M_{j,1})(a)\right] + C2M_1 \qquad \frac{\partial b_1}{\partial a} = V2M_{j,1}$$

$$b_2 = \sum_{j=1}^{6}\left[(V2M_{j,2})(a)\right] + C2M_2 \qquad \frac{\partial b_2}{\partial a} = V2M_{j,2}$$

Therefore, there are two possible routes to calculate V1M.

$$a = \Gamma(h) \qquad \frac{\partial a}{\partial h} = \Gamma'(h) = a'$$

$$h = (V1M)(U_M) + C1M \qquad \frac{\partial h}{\partial V1M} = U_M$$

Therefore,

$$\frac{\partial e}{\partial V1M} = [2(e)(U_s)(V2M_{j,1} \otimes a')](U_M) \quad \text{or:}$$

$$\frac{\partial e}{\partial V1M} = [2(e)(V2M_{j,2} \otimes a')](U_M)$$

In this work, the $V1M_{new}$ is set to the average value of the V1M calculated from each route, as shown in Equation 4.4.

The derivation of the update equation for C1M is as follows:

$$C1M_{new} = C1M - \mu \frac{\partial e}{\partial C1M}$$

$$\frac{\partial e}{\partial C1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial C1M}$$

The partial derivatives are basically the same as in the derivation of the update equations for V1M, except for:

$$\frac{\partial h}{\partial C1M} = 1$$

Therefore, there are also two routes to calculate C1M as given in the following two equations:

$$\frac{\partial e}{\partial C1M} = [2(e)(U_s)(V2M_{j,1} \otimes a')] \quad \text{or:} \quad \frac{\partial e}{\partial C1M} = [2(e)(V2M_{j,2} \otimes a')]$$

Similar to V1M, and C1Mnew is taken as the average as shown in Equation 4.7.

The derivation of the update equation for V2M is as follows:

$$V2M_{new} = V2M - \mu \frac{\partial e}{\partial V2M}$$

$$\frac{\partial e}{\partial V2M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial V2M}$$

$$b = (V2M)(a) + C2M \qquad \frac{\partial b}{\partial V2M} = a$$

Therefore,

$$\frac{\partial e}{\partial V2M} = 2(e)(\underline{U_s})(a^T)$$

where $$\underline{U_s} = \begin{bmatrix} U_s \\ 1 \end{bmatrix}$$

The derivation of the update equation for C2M is as follows:

$$C2M_{new} = C2M - \mu \frac{\partial e}{\partial C2M}$$

$$\frac{\partial e}{\partial C2M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial C2M} = 2(e)(\underline{U_s})$$

B.2 Update Equations for Nonlinear-nonlinear HSNN

The derivation given here is for the nonlinear-nonlinear HSNN structure shown in Figure 4.3. Similar to linear-nonlinear HSNN, four types of parameters must also be derived:

- weights of the first hidden layer of the master network, V1M,
- the biases of the first hidden nodes of the master network, C1M,
- the weights of the second hidden layer of the master network, V2M, and
- the biases of the output nodes of the master network, C2M.

Because of the structure of the network, the derivation for the update of V1M can be obtained through four routes.

Route 1:

$$\frac{\partial e}{\partial V1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{1-3}}\frac{\partial b_{1-3}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial V1M}$$

$$e = (y - y_d)^2, \qquad \frac{\partial e}{\partial y} = 2(y - y_d) = 2e$$

$$y = (V2S)(g) + C2S \qquad \frac{\partial y}{\partial g} = V2S$$

$$g = \Gamma(d) \qquad \frac{\partial g}{\partial d} = \Gamma'(d) = g'$$

$$b_{1-3} = V1S = (V2M_{j,1-3})(a) + C2M_{1-3} \qquad \frac{\partial b_{1-3}}{\partial a} = V2M_{j,1-3}$$

$$d = (V1S)(U_s) + C1S \qquad \frac{\partial d}{\partial b_{1-3}} = U_s$$

$$a = \Gamma(h) \qquad \frac{\partial a}{\partial h} = \Gamma'(h) = a'$$

$$h = (V1M)(U_M) + C1M \qquad \frac{\partial h}{\partial V1M} = U_M$$

$$\frac{\partial e}{\partial V1M} = 2(e)(U_s)(V2S \otimes g')(V2M_{j,1-3} \otimes a')U_M$$

Route 2:

$$\frac{\partial e}{\partial V1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{4-6}}\frac{\partial b_{4-6}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial V1M}$$

$$b_{4-6} = C1S = (V2M_{j,4-6})(a) + C2M_{4-6} \qquad \frac{\partial b_{4-6}}{\partial a} = V2M_{j,4-6}$$

$$d = (V1S)(U_s) + C1S \qquad \frac{\partial d}{\partial b_{4-6}} = 1$$

$$\frac{\partial e}{\partial V1M} = 2(e)(V2S \otimes g')(V2M_{j,4-6} \otimes a')U_M$$

Route 3:

$$\frac{\partial e}{\partial V1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{7-9}}\frac{\partial b_{7-9}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial V1M}$$

$$b_{7-9} = V2S = (V2M_{j,7-9})(a) + C2M_{7-9} \qquad \frac{\partial b_{7-9}}{\partial a} = V2M_{j,7-9}$$

$$y = (V2S)(g) + C2S \qquad \frac{\partial y}{\partial b_{7-9}} = g$$

$$\frac{\partial e}{\partial V1M} = 2(e)(g)(V2M_{j,7-9} \otimes a')U_M$$

Route 4:

$$\frac{\partial e}{\partial V1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{10}}\frac{\partial b_{10}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial V1M}$$

$$b_{10} = C2S = (V2M_{j,10})(a) + C2M_{10} \qquad \frac{\partial b_{10}}{\partial a} = V2M_{j,10}$$

$$y = (V2S)(g) + C2S \qquad \frac{\partial y}{\partial b_{10}} = 1$$

$$\frac{\partial e}{\partial V1M} = 2(e)(V2M_{j,10} \otimes a')U_M$$

The updated V1M can then be calculated from Equation 4.19.

The derivation of the update equation for C1M also has 4 routes.

Route 1:

$$\frac{\partial e}{\partial C1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{1-3}}\frac{\partial b_{1-3}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial C1M}$$

$$h = (V1M)(U_M) + C1M \qquad \frac{\partial h}{\partial C1M} = 1$$

$$\frac{\partial e}{\partial C1M} = 2(e)(U_s)(V2S \otimes g')(V2M_{j,1-3} \otimes a')$$

Route 2:

$$\frac{\partial e}{\partial C1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{4-6}}\frac{\partial b_{4-6}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial C1M}$$

$$\frac{\partial e}{\partial C1M} = 2(e)(V2S \otimes g')(V2M_{j,4-6} \otimes a')$$

Route 3:

$$\frac{\partial e}{\partial C1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{7-9}}\frac{\partial b_{7-9}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial C1M}$$

$$\frac{\partial e}{\partial C1M} = 2(e)(g)(V2M_{j,7-9} \otimes a')$$

Route 4:

$$\frac{\partial e}{\partial C1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{10}}\frac{\partial b_{10}}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial C1M}$$

$$\frac{\partial e}{\partial C1M} = 2(e)(V2M_{j,10} \otimes a')$$

The updated C1M can then be calculated from Equation 4.23.

The update equations for V2M are divided into four groups, according to the output nodes of the master network. The full update formulas for V2M are in Equations 4.11 to 4.14. The derivation for the update formula of V2M ending at output nodes 1 to 3 of the master network is as follows:

$$\frac{\partial e}{\partial V2M_{j,1-3}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{1-3}}\frac{\partial b_{1-3}}{\partial V2M_{j,1-3}}$$

$$b_{1-3} = V1S = (V2M_{j,1-3})(a) + C2M_{1-3} \qquad \frac{\partial b_{1-3}}{\partial V2M_{j,1-3}} = a$$

$$\frac{\partial e}{\partial V2M_{j,1-3}} = 2(e)(U_s)(V2S \otimes g')(a)$$

The derivation for the update formula of V2M ending at the master output nodes 4 to 6 is as follows:

$$\frac{\partial e}{\partial V2M_{j,4-6}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{4-6}}\frac{\partial b_{4-6}}{\partial V2M_{j,4-6}}$$

$$b_{4-6} = C1S = (V2M_{j,4-6})(a) + C2M_{4-6} \qquad \frac{\partial b_{4-6}}{\partial V2M_{j,4-6}} = a$$

$$\frac{\partial e}{\partial V2M_{j,4-6}} = 2(e)(V2S \otimes g')(a)$$

The derivation for the update formula of V2M ending at the master output nodes 7 to 9 is as follows:

$$\frac{\partial e}{\partial V2M_{j,7-9}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{7-9}}\frac{\partial b_{7-9}}{\partial V2M_{j,7-9}}$$

$$b_{7-9} = V2S = (V2M_{j,7-9})(a) + C2M_{7-9} \qquad \frac{\partial b_{7-9}}{\partial V2M_{j,7-9}} = a$$

$$\frac{\partial e}{\partial V2M_{j,7\text{-}9}} = 2(e)(g)(a)$$

The derivation for the update formula of V2M ending at the master output node 10 is as follows:

$$\frac{\partial e}{\partial V2M_{j,10}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{10}}\frac{\partial b_{10}}{\partial V2M_{j,10}}$$

$$b_{10} = C2S = (V2M_{j,10})(a) + C2M_{10} \qquad \frac{\partial b_{10}}{\partial V2M_{j,10}} = a$$

$$\frac{\partial e}{\partial V2M_{j,10}} = 2(e)(a)$$

The full update formulas for C2M are in Equations 4.15 to 4.18. The derivation for the update formula of C2M ending at the master output node 1 to 3 is as follows:

$$\frac{\partial e}{\partial C2M_{j,1\text{-}3}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{1\text{-}3}}\frac{\partial b_{1\text{-}3}}{\partial C2M_{j,1\text{-}3}}$$

$$b_{1\text{-}3} = V1S = (V2M_{j,1\text{-}3})(a) + C2M_{1\text{-}3} \qquad \frac{\partial b_{1\text{-}3}}{\partial C2M_{1\text{-}3}} = 1$$

$$\frac{\partial e}{\partial C2M_{j,1\text{-}3}} = 2(e)(U_s)(V2S \otimes g')$$

The derivation for the update formula of C2M ending at the master output node 4 to 6 is as follows:

$$\frac{\partial e}{\partial C2M_{j,4\text{-}6}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial b_{4\text{-}6}}\frac{\partial b_{4\text{-}6}}{\partial C2M_{j,4\text{-}6}}$$

$$b_{4\text{-}6} = C1S = (V2M_{j,4\text{-}6})(a) + C2M_{4\text{-}6} \qquad \frac{\partial b_{4\text{-}6}}{\partial C2M_{j,4\text{-}6}} = 1$$

$$\frac{\partial e}{\partial C2M_{j,4-6}} = 2(e)(V2S \otimes g')$$

The derivation for the update formula of C2M ending at the master output node 7 to 9 is as follows:

$$\frac{\partial e}{\partial C2M_{j,7-9}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{7-9}}\frac{\partial b_{7-9}}{\partial C2M_{j,7-9}}$$

$$b_{7-9} = V2S = (V2M_{j,7-9})(a) + C2M_{7-9} \qquad \frac{\partial b_{7-9}}{\partial C2M_{j,7-9}} = 1$$

$$\frac{\partial e}{\partial C2M_{j,7-9}} = 2(e)(g)$$

The derivation for the update formula of V2M ending at the master output node 10 is as follows:

$$\frac{\partial e}{\partial C2M_{j,10}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_{10}}\frac{\partial b_{10}}{\partial C2M_{j,10}}$$

$$b_{10} = C2S = (V2M_{j,10})(a) + C2M_{10} \qquad \frac{\partial b_{10}}{\partial C2M_{j,10}} = 1$$

$$\frac{\partial e}{\partial C2M_{j,10}} = 2(e)$$

B.3 Update Equations for Output-tuned HSNN

The derivation given here is for the nonlinear-nonlinear HSNN structure shown in Figure 4.4. There are 8 types of parameters to be derived:

- the weights of the first hidden layer of the master network, V1M,
- the biases of the first hidden nodes of the master network, C1M,
- the weights of the second hidden layer of the master network, V2M,
- the biases of the output nodes of the master network, C2M,
- the weights of the first hidden layer of the slave network, V1S,
- the biases of the first hidden nodes of the slave network, C1S,
- the weights of the second hidden layer of the slave network, V2S, and
- the biases of the output nodes of the slave network, C2S.

Because of the structure of the master network, the derivation for the update of V1M can be obtained through two routes.

Route 1:

$$\frac{\partial e}{\partial V1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_1}\frac{\partial b_1}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial V1M}$$

$$e = (y - y_d)^2, \qquad \frac{\partial e}{\partial y} = 2(y - y_d) = 2e$$

$$y = (y_s)(b_1) + b_2 \qquad \frac{\partial y}{\partial b_1} = y_s$$

$$b_1 = \sum_{j=1}^{6}\left[(V2M_{j,1})(a)\right] + C2M_1 \qquad \frac{\partial b_1}{\partial a} = V2M_{j,1}$$

$$a = \Gamma(h) \qquad \frac{\partial a}{\partial h} = \Gamma'(h) = a'$$

$$h = (V1M)(U_M) + C1M \qquad \frac{\partial h}{\partial V1M} = U_M$$

$$\frac{\partial e}{\partial V1M} = [2(e)(y_s)(V2M_{j,1} \otimes a')](U_M)$$

Route 2:

$$\frac{\partial y}{\partial b_2} = 1$$

$$b_2 = \sum_{j=1}^{6}\left[(V2M_{j,2})(a)\right] + C2M_2 \qquad \frac{\partial b_2}{\partial a} = V2M_{j,2}$$

$$\frac{\partial e}{\partial V1M} = [2(e)(V2M_{j,2} \otimes a')](U_M)$$

In this work, the $V1M_{new}$ is set to the average value of the V1M calculated from each route, as shown in Equation 4.33.

The derivation of the update equation for C1M is as follows:

$$\frac{\partial e}{\partial C1M} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial C1M}$$

The partial derivatives are basically the same as in the derivation of the update equations for V1M, except for:

$$\frac{\partial h}{\partial C1M} = 1$$

Therefore, there are also two routes to calculate C1M as given in the following two equations:

$$\frac{\partial e}{\partial C1M} = [2(e)(y_s)(V2M_{j,1} \otimes a')] \quad \text{or:} \quad \frac{\partial e}{\partial C1M} = [2(e)(V2M_{j,2} \otimes a')]$$

Similar to V1M, and $C1M_{new}$ is taken as the average as shown in Equation 4.36.

The update equations for V2M are divided into two groups, according to the output nodes of the master network. The full update formulas for V2M are in Equations 4.29 and 4.30. The derivation for the update formula of V2M ending at output node 1 of the master network is as follows:

$$\frac{\partial e}{\partial V2M_{j,1}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_1}\frac{\partial b_1}{\partial V2M_{j,1}}$$

$$b_1 = \sum_{j=1}^{6}\left[(V2M_{j,1})(a)\right] + C2M_1 \qquad \frac{\partial b_1}{\partial V2M_{j,1}} = a$$

$$\frac{\partial e}{\partial V2M_{j,1}} = 2(e)(y_s)(a)$$

The derivation for the update formula of V2M ending at output node 2 of the master network is as follows:

$$\frac{\partial e}{\partial V2M_{j,2}} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_2}\frac{\partial b_2}{\partial V2M_{j,2}}$$

$$b_2 = \sum_{j=1}^{6}\left[(V2M_{j,2})(a)\right] + C2M_2 \qquad \frac{\partial b_2}{\partial V2M_{j,2}} = a$$

$$\frac{\partial e}{\partial V2M_{j,2}} = 2(e)(a)$$

The full update formulas for C2M are in Equations 4.31 and 4.32. The derivation for the update formula of C2M ending at the master output node 1 is as follows:

$$\frac{\partial e}{\partial C2M_1} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_1}\frac{\partial b_1}{\partial C2M_1}$$

$$b_1 = \sum_{j=1}^{6}\left[(V2M_{j,1})(a)\right] + C2M_1 \qquad \frac{\partial b_1}{\partial C2M_1} = 1$$

$$\frac{\partial e}{\partial C2M_1} = 2(e)(y_s)$$

The derivation for the update formula of C2M ending at the master output node 2 is as follows:

$$\frac{\partial e}{\partial C2M_2} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial b_2}\frac{\partial b_2}{\partial C2M_2}$$

$$b_2 = \sum_{j=1}^{6}\left[(V2M_{j,2})(a)\right] + C2M_2 \qquad \frac{\partial b_2}{\partial C2M_2} = 1$$

$$\frac{\partial e}{\partial C2M_2} = 2(e)$$

The full update formulas for the slave network are in Equations 4.39 to 4.42. The derivation for the update equation of V1S is as follows:

$$\frac{\partial e}{\partial V1S} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial y_s}\frac{\partial y_s}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial V1S}$$

$$d = (V1S)(U_s) + C1S \qquad \frac{\partial d}{\partial V1S} = U_s$$

$$g = \Gamma(d) \qquad \frac{\partial g}{\partial d} = \Gamma'(d) = g'$$

$$y_s = (V2S)(g) + C2S \qquad \frac{\partial y_s}{\partial g} = V2S$$

$$y = (y_s)(b_1) + b_2 \qquad \frac{\partial y}{\partial y_s} = b_1$$

$$\frac{\partial e}{\partial V1S} = [2(e)(b_1)(V2S \otimes a')](U_s)$$

The derivation for the update equation of C1S is as follows:

$$\frac{\partial e}{\partial C1S} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial y_s}\frac{\partial y_s}{\partial g}\frac{\partial g}{\partial d}\frac{\partial d}{\partial C1S}$$

$$d = (V1S)(U_s) + C1S \qquad \frac{\partial d}{\partial C1S} = 1$$

$$\frac{\partial e}{\partial C1S} = [2(e)(b_1)(V2S \otimes a')]$$

The derivation for the update equation of V2S is as follows:

$$\frac{\partial e}{\partial V2S} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial y_s}\frac{\partial y_s}{\partial V2S}$$

$$y_s = (V2S)(g) + C2S \qquad \frac{\partial y_s}{\partial V2S} = g$$

$$\frac{\partial e}{\partial V2S} = 2(e)(b_1)(g)$$

The derivation for the update equation of C2S is as follows:

$$\frac{\partial e}{\partial C2S} = \frac{\partial e}{\partial y}\frac{\partial y}{\partial y_s}\frac{\partial y_s}{\partial C2S}$$

$$y_s = (V2S)(g) + C2S \qquad \frac{\partial y_s}{\partial C2S} = 1$$

$$\frac{\partial e}{\partial C2S} = 2(e)(b_1)$$

APPENDIX C
TRAINING AND TEST DATA

C.1 TRAINING AND TESTING DATA FOR THE FLASH SYSTEM

All training and testing data given are normalised.

Main training data for M-W flash

File name: flth815

Number of data points: 150

T	P	F	z	K	V	y	L	x	Q
0	0	0.76	0.473684	0.415409	0.552345	0.647154	0.447655	0.259646	0.390285
0	0.125	0.6	0.333333	0.362947	0.047557	0.687381	0.952443	0.315655	0.038138
0	0.125	1	0.6	0.362952	0.764939	0.687381	0.235061	0.315644	0.699536
0	0.25	0.76	0.473684	0.320358	0.278329	0.724529	0.721671	0.37694	0.202883
0	0.375	0.52	0.230769	0.355572	0	0	1	0.230769	0.010691
0	0.375	1	0.6	0.2861	0.49724	0.759378	0.50276	0.442372	0.457677
0	0.5	0.76	0.473684	0.264512	0	0	1	0.473684	0.017752
0	0.625	0.52	0.230769	0.318144	0	0	1	0.230769	0.010691
0	0.625	1	0.6	0.236735	0.080569	0.824302	0.919431	0.580344	0.094091
0	0.75	0.76	0.473684	0.238061	0	0	1	0.473684	0.017752
0	0.875	0.52	0.230769	0.287844	0	0	1	0.230769	0.010691
0	0.875	0.92	0.565217	0.215637	0	0	1	0.565217	0.022428
0	1	0.76	0.473684	0.216419	0	0	1	0.473684	0.017752
0.1	0	0.52	0.230769	0.48123	0.061214	0.597285	0.938785	0.20687	0.041131
0.1	0	0.92	0.565217	0.481243	0.917881	0.597278	0.082119	0.206853	0.781746
0.1	0.125	0.68	0.411765	0.421326	0.411495	0.639092	0.588505	0.252813	0.266124
0.1	0.25	0.52	0.230769	0.406864	0	0	1	0.230769	0.012859

0.1	0.25	0.92	0.565217	0.371393	0.69997	0.677222	0.30003	0.303911	0.59391
0.1	0.375	0.68	0.411765	0.330116	0.147443	0.71254	0.852557	0.359748	0.106081
0.1	0.5	0.52	0.230769	0.361657	0	0	1	0.230769	0.012859
0.1	0.5	0.92	0.565217	0.296296	0.446742	0.745718	0.553258	0.419467	0.384284
0.1	0.625	0.68	0.411765	0.282398	0	0	1	0.411765	0.018532
0.1	0.75	0.44	0.090909	0.413191	0	0	1	0.090909	0.010015
0.1	0.75	0.92	0.565217	0.24633	0.072037	0.807551	0.927963	0.546405	0.083954
0.1	0.875	0.68	0.411765	0.255503	0	0	1	0.411765	0.018532
0.1	1	0.44	0.090909	0.375628	0	0	1	0.090909	0.010015
0.1	1	0.92	0.565217	0.221945	0	0	1	0.565217	0.026982
0.2	0	0.68	0.411765	0.553631	0.649838	0.545202	0.350162	0.164129	0.418398
0.2	0.125	0.44	0.090909	0.591814	0	0	1	0.090909	0.011709
0.2	0.125	0.84	0.52381	0.486886	0.831339	0.589163	0.168662	0.201678	0.65096
0.2	0.25	0.68	0.411765	0.430179	0.436462	0.628842	0.563538	0.243638	0.283872
0.2	0.375	0.44	0.090909	0.522189	0	0	1	0.090909	0.011709
0.2	0.375	0.84	0.52381	0.382204	0.623131	0.665178	0.376868	0.290064	0.488263
0.2	0.5	0.68	0.411765	0.341871	0.198229	0.698921	0.801771	0.340768	0.139232
0.2	0.625	0.44	0.090909	0.467221	0	0	1	0.090909	0.011709
0.2	0.625	0.84	0.52381	0.308272	0.383769	0.730624	0.616231	0.395012	0.308147
0.2	0.75	0.6	0.333333	0.309985	0	0	1	0.333333	0.018361
0.2	0.875	0.44	0.090909	0.422724	0	0	1	0.090909	0.011709
0.2	0.875	0.84	0.52381	0.257397	0.044736	0.789699	0.955264	0.511358	0.060547
0.2	1	0.6	0.333333	0.281804	0	0	1	0.333333	0.018361
0.2	1	1	0.6	0.238381	0.11509	0.817663	0.88491	0.571691	0.133221
0.3	0	0.84	0.52381	0.608333	1	0.52381	0	0	0.788625
0.3	0.125	0.6	0.333333	0.558716	0.459471	0.537017	0.540529	0.160194	0.267731
0.3	0.125	1	0.6	0.5131	1	0.6	0	0	0.927478
0.3	0.25	0.84	0.52381	0.495794	0.85698	0.578758	0.14302	0.194556	0.673408
0.3	0.375	0.6	0.333333	0.44166	0.262119	0.616633	0.737881	0.232696	0.159602
0.3	0.375	1	0.6	0.441662	0.956683	0.616631	0.043317	0.232693	0.886252
0.3	0.5	0.76	0.473684	0.39523	0.528176	0.651431	0.471824	0.274709	0.380808
0.3	0.625	0.6	0.333333	0.355617	0.035379	0.683792	0.964621	0.32048	0.039507
0.3	0.625	1	0.6	0.355621	0.769374	0.683792	0.230625	0.320468	0.711721
0.3	0.75	0.76	0.473684	0.322047	0.301958	0.714226	0.698042	0.369631	0.227611
0.3	0.875	0.6	0.333333	0.31752	0	0	1	0.333333	0.021034
0.3	0.875	1	0.6	0.293768	0.554806	0.743142	0.445194	0.421615	0.51893
0.3	1	0.76	0.473684	0.270405	0	0	1	0.473684	0.028616
0.4	0	0.52	0.230769	0.715515	0.391996	0.43246	0.608004	0.100734	0.204745
0.4	0	1	0.6	0.5868	1	0.6	0	0	0.929494
0.4	0.125	0.76	0.473684	0.636164	0.976181	0.48216	0.023819	0.126319	0.703173
0.4	0.25	0.52	0.230769	0.567386	0.204788	0.526416	0.795212	0.154632	0.114362
0.4	0.25	1	0.6	0.51449	1	0.6	0	0	0.929494
0.4	0.375	0.76	0.473684	0.50758	0.756506	0.566296	0.243494	0.185947	0.54339
0.4	0.5	0.52	0.230769	0.455528	0.026851	0.602644	0.973148	0.220508	0.031685
0.4	0.5	0.92	0.565217	0.455546	0.902087	0.602635	0.097913	0.220481	0.774146
0.4	0.625	0.76	0.473684	0.410355	0.569997	0.636121	0.430003	0.258363	0.412104

0.4	0.75	0.52	0.230769	0.404194	0	0	1	0.230769	0.019423
0.4	0.75	0.92	0.565217	0.371268	0.722404	0.667301	0.277596	0.29956	0.620636
0.4	0.875	0.68	0.411765	0.337621	0.192323	0.696637	0.807677	0.343931	0.141263
0.4	1	0.52	0.230769	0.367449	0	0	1	0.230769	0.019423
0.4	1	0.92	0.565217	0.308875	0.522559	0.724468	0.477441	0.390918	0.455636
0.5	0	0.68	0.411765	0.778524	1	0.411765	0	0	0.652593
0.5	0.125	0.52	0.230769	0.718846	0.40643	0.424168	0.59357	0.098345	0.213526
0.5	0.125	0.92	0.565217	0.614644	1	0.565217	0	0	0.861706
0.5	0.25	0.68	0.411765	0.64437	0.829491	0.471346	0.17051	0.121914	0.540237
0.5	0.375	0.44	0.090909	0.643704	0	0	1	0.090909	0.016839
0.5	0.375	0.92	0.565217	0.543149	1	0.565217	0	0	0.861706
0.5	0.5	0.68	0.411765	0.521958	0.626877	0.551949	0.373123	0.176244	0.40955
0.5	0.625	0.44	0.090909	0.575945	0	0	1	0.090909	0.016839
0.5	0.625	0.92	0.565217	0.471654	0.942632	0.586992	0.057368	0.207424	0.811475
0.5	0.75	0.68	0.411765	0.427477	0.450636	0.619358	0.549364	0.241479	0.299531
0.5	0.875	0.44	0.090909	0.521093	0	0	1	0.090909	0.016839
0.5	0.875	0.84	0.52381	0.388792	0.661219	0.649527	0.338782	0.27844	0.524972
0.5	1	0.68	0.411765	0.355054	0.260077	0.677904	0.739923	0.318219	0.184317
0.6	0	0.44	0.090909	0.898453	0.137494	0.305576	0.862506	0.056688	0.074592
0.6	0	0.84	0.52381	0.741196	1	0.52381	0	0	0.793587
0.6	0.125	0.68	0.411765	0.775749	1	0.411765	0	0	0.653883
0.6	0.25	0.44	0.090909	0.732013	0	0	1	0.090909	0.018565
0.6	0.25	0.84	0.52381	0.649142	1	0.52381	0	0	0.793587
0.6	0.375	0.6	0.333333	0.65596	0.634599	0.458227	0.365401	0.116428	0.372077
0.6	0.5	0.44	0.090909	0.650678	0	0	1	0.090909	0.018565
0.6	0.5	0.84	0.52381	0.576993	1	0.52381	0	0	0.793587
0.6	0.625	0.6	0.333333	0.538696	0.452728	0.535864	0.547272	0.165791	0.269828
0.6	0.625	1	0.6	0.494408	1	0.6	0	0	0.933538
0.6	0.75	0.84	0.52381	0.489841	0.87764	0.569808	0.12236	0.193876	0.695742
0.6	0.875	0.6	0.333333	0.446506	0.289037	0.601223	0.710963	0.224424	0.180772
0.6	0.875	1	0.6	0.446511	0.996753	0.601223	0.003247	0.224415	0.930467
0.6	1	0.84	0.52381	0.408147	0.713914	0.630536	0.286086	0.257479	0.567812
0.7	0	0.6	0.333333	0.941388	1	0.333333	0	0	0.585388
0.7	0	1	0.6	0.714562	1	0.6	0	0	0.935568
0.7	0.125	0.76	0.473684	0.780426	1	0.473684	0	0	0.72516
0.7	0.25	0.6	0.333333	0.813352	0.935165	0.35145	0.064834	0.072017	0.547449
0.7	0.25	1	0.6	0.625242	1	0.6	0	0	0.935568
0.7	0.375	0.76	0.473684	0.688896	1	0.473684	0	0	0.72516
0.7	0.5	0.6	0.333333	0.670549	0.670567	0.442998	0.329433	0.110108	0.394261
0.7	0.5	1	0.6	0.555771	1	0.6	0	0	0.935568
0.7	0.625	0.76	0.473684	0.610876	0.975521	0.482269	0.024479	0.131578	0.707256
0.7	0.75	0.52	0.230769	0.557614	0.208933	0.518143	0.791067	0.154869	0.122139
0.7	0.75	1	0.6	0.501143	1	0.6	0	0	0.935568
0.7	0.875	0.76	0.473684	0.509985	0.791177	0.551166	0.208823	0.180125	0.574024
0.7	1	0.52	0.230769	0.467364	0.062223	0.581782	0.937777	0.207479	0.054315
0.7	1	1	0.6	0.455564	1	0.6	0	0	0.935568

0.8	0	0.76	0.473684	0.891167	1	0.473684	0	0	0.72664
0.8	0.125	0.52	0.230769	0.995713	1	0.230769	0	0	0.516934
0.8	0.125	0.92	0.565217	0.746184	1	0.565217	0	0	0.867241
0.8	0.25	0.76	0.473684	0.779771	1	0.473684	0	0	0.72664
0.8	0.375	0.52	0.230769	0.82405	0.603305	0.337611	0.396695	0.068282	0.316043
0.8	0.375	0.92	0.565217	0.658398	1	0.565217	0	0	0.867241
0.8	0.5	0.68	0.411765	0.735039	1	0.411765	0	0	0.65647
0.8	0.625	0.52	0.230769	0.68782	0.395466	0.425811	0.604534	0.103179	0.21322
0.8	0.625	0.92	0.565217	0.589943	1	0.565217	0	0	0.867241
0.8	0.75	0.68	0.411765	0.630279	0.847147	0.463925	0.152853	0.122677	0.556488
0.8	0.875	0.52	0.230769	0.578563	0.245133	0.498865	0.754867	0.143709	0.141106
0.8	0.875	0.92	0.565217	0.533758	1	0.565217	0	0	0.867241
0.8	1	0.68	0.411765	0.531986	0.672774	0.53111	0.327226	0.166392	0.444481
0.9	0	0.44	0.090909	1.208998	1	0.090909	0	0	0.449068
0.9	0	0.92	0.565217	0.851549	1	0.565217	0	0	0.869094
0.9	0.125	0.68	0.411765	0.939276	1	0.411765	0	0	0.657768
0.9	0.25	0.44	0.090909	1.001984	0.304681	0.216276	0.695319	0.035975	0.149536
0.9	0.25	0.92	0.565217	0.745105	1	0.565217	0	0	0.869094
0.9	0.375	0.68	0.411765	0.828865	1	0.411765	0	0	0.657768
0.9	0.5	0.44	0.090909	0.838198	0.104902	0.321386	0.895098	0.063898	0.06601
0.9	0.5	0.84	0.52381	0.697938	1	0.52381	0	0	0.79858
0.9	0.625	0.68	0.411765	0.741612	1	0.411765	0	0	0.657768
0.9	0.75	0.44	0.090909	0.714296	0	0	1	0.090909	0.02379
0.9	0.75	0.84	0.52381	0.628713	1	0.52381	0	0	0.79858
0.9	0.875	0.68	0.411765	0.651809	0.902654	0.443929	0.097346	0.113513	0.594117
0.9	1	0.44	0.090909	0.64936	0	0	1	0.090909	0.02379
0.9	1	0.84	0.52381	0.571538	1	0.52381	0	0	0.79858
1	0	0.6	0.333333	1.137829	1	0.333333	0	0	0.588723
1	0.125	0.44	0.090909	1.201687	1	0.090909	0	0	0.449809
1	0.125	0.84	0.52381	0.891267	1	0.52381	0	0	0.800252
1	0.25	0.6	0.333333	0.995601	1	0.333333	0	0	0.588723
1	0.25	1	0.6	0.756078	1	0.6	0	0	0.941684
1	0.375	0.84	0.52381	0.786412	1	0.52381	0	0	0.800252
1	0.5	0.6	0.333333	0.885002	1	0.333333	0	0	0.588723
1	0.5	1	0.6	0.672069	1	0.6	0	0	0.941684
1	0.625	0.84	0.52381	0.703632	1	0.52381	0	0	0.800252
1	0.75	0.6	0.333333	0.789231	0.952228	0.346386	0.047772	0.073149	0.56097
1	0.75	1	0.6	0.604862	1	0.6	0	0	0.941684
1	0.875	0.76	0.473684	0.673547	1	0.473684	0	0	0.729609
1	1	0.6	0.333333	0.675326	0.720192	0.422343	0.279808	0.104232	0.427342
1	1	1	0.6	0.549874	1	0.6	0	0	0.941684

Test data for M-W system

File name: fts70k

Number of data points: 70, error in K-values for single-phase region

T	P	F	z	K	V	y	L	x	Q
0.075	0.18	0.66	0.393939	0.383686	0.274136	0.668322	0.725864	0.290314	0.176409
0.075	0.18	0.92	0.565217	0.383692	0.727259	0.668318	0.272741	0.290302	0.616376
0.075	0.355	0.66	0.393939	0.324137	0.06968	0.7188	0.93032	0.369608	0.057213
0.075	0.355	0.92	0.565217	0.324142	0.560198	0.7188	0.439802	0.369592	0.476506
0.075	0.53	0.66	0.393939	0.292956	0	0	1	0.393939	0.017072
0.075	0.53	0.92	0.565217	0.279381	0.352456	0.765096	0.647544	0.456424	0.307009
0.075	0.705	0.66	0.393939	0.272074	0	0	1	0.393939	0.017072
0.075	0.705	0.92	0.565217	0.246027	0.06701	0.808528	0.93299	0.547742	0.07888
0.075	0.75	0.66	0.393939	0.267176	0	0	1	0.393939	0.017072
0.075	0.75	0.92	0.565217	0.239606	0	0	1	0.565217	0.025842
0.215	0.18	0.66	0.393939	0.470752	0.469086	0.599554	0.530914	0.21227	0.296105
0.215	0.18	0.92	0.565217	0.470753	0.91134	0.599554	0.088659	0.212268	0.778366
0.215	0.355	0.66	0.393939	0.397963	0.318456	0.652339	0.681544	0.2732	0.206017
0.215	0.355	0.92	0.565217	0.397964	0.770213	0.652339	0.229787	0.273199	0.65643
0.215	0.53	0.66	0.393939	0.340543	0.144218	0.699645	0.855782	0.342421	0.104235
0.215	0.53	0.92	0.565217	0.340546	0.623696	0.699645	0.376304	0.342414	0.533187
0.215	0.705	0.66	0.393939	0.301801	0	0	1	0.393939	0.021301
0.215	0.705	0.92	0.565217	0.295889	0.451895	0.743092	0.548105	0.418565	0.392277
0.215	0.75	0.66	0.393939	0.296368	0	0	1	0.393939	0.021301
0.215	0.75	0.92	0.565217	0.286167	0.400928	0.753789	0.599072	0.439016	0.351041
0.355	0.18	0.66	0.393939	0.57035	0.643805	0.526733	0.356194	0.153922	0.406364
0.355	0.18	0.92	0.565217	0.543123	1	0.565217	0	0	0.859044
0.355	0.355	0.66	0.393939	0.485796	0.505965	0.583218	0.494035	0.200091	0.32148
0.355	0.355	0.92	0.565217	0.485796	0.953018	0.583218	0.046982	0.20009	0.817693
0.355	0.53	0.66	0.393939	0.416743	0.370761	0.632917	0.629239	0.253128	0.240325
0.355	0.53	0.92	0.565217	0.416749	0.821748	0.632917	0.178252	0.253117	0.703822
0.355	0.705	0.66	0.393939	0.360732	0.22174	0.677645	0.77826	0.313106	0.152928
0.355	0.705	0.92	0.565217	0.360745	0.691629	0.677638	0.308371	0.313075	0.593658
0.355	0.75	0.66	0.393939	0.348188	0.179248	0.688516	0.820752	0.329605	0.128314
0.355	0.75	0.92	0.565217	0.348208	0.656537	0.688506	0.343463	0.329549	0.564402
0.495	0.18	0.66	0.393939	0.680702	0.838896	0.448504	0.161104	0.109814	0.531834
0.495	0.18	0.92	0.565217	0.596061	1	0.565217	0	0	0.861614
0.495	0.355	0.66	0.393939	0.585304	0.682093	0.509876	0.317907	0.145188	0.432819
0.495	0.355	0.92	0.565217	0.546508	1	0.565217	0	0	0.861614
0.495	0.53	0.66	0.393939	0.505892	0.551814	0.563198	0.448186	0.185546	0.352315
0.495	0.53	0.92	0.565217	0.504542	1	0.565217	0	0	0.861614
0.495	0.705	0.66	0.393939	0.439664	0.428815	0.610436	0.571185	0.231406	0.278084
0.495	0.705	0.92	0.565217	0.439667	0.880701	0.610436	0.119299	0.231401	0.757656

0.495	0.75	0.66	0.393939	0.42451	0.396687	0.621788	0.603313	0.244126	0.258974
0.495	0.75	0.92	0.565217	0.424514	0.850211	0.621788	0.149789	0.244118	0.731424
0.635	0.18	0.66	0.393939	0.782482	1	0.393939	0	0	0.636894
0.635	0.18	0.92	0.565217	0.652153	1	0.565217	0	0	0.864192
0.635	0.355	0.66	0.393939	0.694964	0.886732	0.431055	0.113269	0.103376	0.564388
0.635	0.355	0.92	0.565217	0.59882	1	0.565217	0	0	0.864192
0.635	0.53	0.66	0.393939	0.605871	0.731744	0.489039	0.268256	0.134528	0.466349
0.635	0.53	0.92	0.565217	0.552858	1	0.565217	0	0	0.864192
0.635	0.705	0.66	0.393939	0.530456	0.605855	0.53987	0.394145	0.169625	0.388201
0.635	0.705	0.92	0.565217	0.513423	1	0.565217	0	0	0.864192
0.635	0.75	0.66	0.393939	0.512978	0.575873	0.55199	0.424127	0.179342	0.369836
0.635	0.75	0.92	0.565217	0.504181	1	0.565217	0	0	0.864192
0.775	0.18	0.66	0.393939	0.856041	1	0.393939	0	0	0.638641
0.775	0.18	0.92	0.565217	0.71349	1	0.565217	0	0	0.866778
0.775	0.355	0.66	0.393939	0.78487	1	0.393939	0	0	0.638641
0.775	0.355	0.92	0.565217	0.654174	1	0.565217	0	0	0.866778
0.775	0.53	0.66	0.393939	0.715442	0.951141	0.409278	0.048859	0.095344	0.607412
0.775	0.53	0.92	0.565217	0.603963	1	0.565217	0	0	0.866778
0.775	0.705	0.66	0.393939	0.631301	0.793319	0.464615	0.206681	0.122661	0.507345
0.775	0.705	0.92	0.565217	0.561723	1	0.565217	0	0	0.866778
0.775	0.75	0.66	0.393939	0.611674	0.758893	0.47774	0.241107	0.130173	0.485731
0.775	0.75	0.92	0.565217	0.551612	1	0.565217	0	0	0.866778
0.9	0.18	0.66	0.393939	0.926269	1	0.393939	0	0	0.640204
0.9	0.18	0.92	0.565217	0.77213	1	0.565217	0	0	0.869094
0.9	0.355	0.66	0.393939	0.849322	1	0.393939	0	0	0.640204
0.9	0.355	0.92	0.565217	0.707938	1	0.565217	0	0	0.869094
0.9	0.53	0.66	0.393939	0.784129	1	0.393939	0	0	0.640204
0.9	0.53	0.92	0.565217	0.653601	1	0.565217	0	0	0.869094
0.9	0.705	0.66	0.393939	0.728235	1	0.393939	0	0	0.640204
0.9	0.705	0.92	0.565217	0.60701	1	0.565217	0	0	0.869094
0.9	0.75	0.66	0.393939	0.707531	0.958695	0.406784	0.041305	0.095822	0.613885
0.9	0.75	0.92	0.565217	0.596084	1	0.565217	0	0	0.869094

Training data for M-W system

File name: flth150.txt

Number of data points: 150 with no error in K-values.

T	P	F	z	K	V	y	L	x	q
0	0	0.76	0.473684	0.41541	0.552345	0.647154	0.44766	0.259646	0.39029
0	0.125	0.6	0.333333	0.36295	0.047557	0.687381	0.95244	0.315655	0.03814

0	0.125	1	0.6	0.36295	0.764939	0.687381	0.23506	0.315644	0.69954
0	0.25	0.76	0.473684	0.32036	0.278329	0.724529	0.72167	0.37694	0.20288
0	0.375	0.52	0.230769	0	0	0	1	0.230769	0.01069
0	0.375	1	0.6	0.2861	0.49724	0.759378	0.50276	0.442372	0.45768
0	0.5	0.76	0.473684	0	0	0	1	0.473684	0.01775
0	0.625	0.52	0.230769	0	0	0	1	0.230769	0.01069
0	0.625	1	0.6	0.23674	0.080569	0.824302	0.91943	0.580344	0.09409
0	0.75	0.76	0.473684	0	0	0	1	0.473684	0.01775
0	0.875	0.52	0.230769	0	0	0	1	0.230769	0.01069
0	0.875	0.92	0.565217	0	0	0	1	0.565217	0.02243
0	1	0.76	0.473684	0	0	0	1	0.473684	0.01775
0.1	0	0.52	0.230769	0.48123	0.061214	0.597285	0.93879	0.20687	0.04113
0.1	0	0.92	0.565217	0.48124	0.917881	0.597278	0.08212	0.206853	0.78175
0.1	0.125	0.68	0.411765	0.42133	0.411495	0.639092	0.58851	0.252813	0.26612
0.1	0.25	0.52	0.230769	0	0	0	1	0.230769	0.01286
0.1	0.25	0.92	0.565217	0.37139	0.69997	0.677222	0.30003	0.303911	0.59391
0.1	0.375	0.68	0.411765	0.33012	0.147443	0.71254	0.85256	0.359748	0.10608
0.1	0.5	0.52	0.230769	0	0	0	1	0.230769	0.01286
0.1	0.5	0.92	0.565217	0.2963	0.446742	0.745718	0.55326	0.419467	0.38428
0.1	0.625	0.68	0.411765	0	0	0	1	0.411765	0.01853
0.1	0.75	0.44	0.090909	0	0	0	1	0.090909	0.01002
0.1	0.75	0.92	0.565217	0.24633	0.072037	0.807551	0.92796	0.546405	0.08395
0.1	0.875	0.68	0.411765	0	0	0	1	0.411765	0.01853
0.1	1	0.44	0.090909	0	0	0	1	0.090909	0.01002
0.1	1	0.92	0.565217	0	0	0	1	0.565217	0.02698
0.2	0	0.68	0.411765	0.55363	0.649838	0.545202	0.35016	0.164129	0.4184
0.2	0.125	0.44	0.090909	0	0	0	1	0.090909	0.01171
0.2	0.125	0.84	0.52381	0.48689	0.831339	0.589163	0.16866	0.201678	0.65096
0.2	0.25	0.68	0.411765	0.43018	0.436462	0.628842	0.56354	0.243638	0.28387
0.2	0.375	0.44	0.090909	0	0	0	1	0.090909	0.01171
0.2	0.375	0.84	0.52381	0.3822	0.623131	0.665178	0.37687	0.290064	0.48826
0.2	0.5	0.68	0.411765	0.34187	0.198229	0.698921	0.80177	0.340768	0.13923
0.2	0.625	0.44	0.090909	0	0	0	1	0.090909	0.01171
0.2	0.625	0.84	0.52381	0.30827	0.383769	0.730624	0.61623	0.395012	0.30815
0.2	0.75	0.6	0.333333	0	0	0	1	0.333333	0.01836
0.2	0.875	0.44	0.090909	0	0	0	1	0.090909	0.01171
0.2	0.875	0.84	0.52381	0.2574	0.044736	0.789699	0.95526	0.511358	0.06055
0.2	1	0.6	0.333333	0	0	0	1	0.333333	0.01836
0.2	1	1	0.6	0.23838	0.11509	0.817663	0.88491	0.571691	0.13322
0.3	0	0.84	0.52381	1	1	0.52381	0	0	0.78863
0.3	0.125	0.6	0.333333	0.55872	0.459471	0.537017	0.54053	0.160194	0.26773
0.3	0.125	1	0.6	1	1	0.6	0	0	0.92748
0.3	0.25	0.84	0.52381	0.49579	0.85698	0.578758	0.14302	0.194556	0.67341
0.3	0.375	0.6	0.333333	0.44166	0.262119	0.616633	0.73788	0.232696	0.1596
0.3	0.375	1	0.6	0.44166	0.956683	0.616631	0.04332	0.232693	0.88625
0.3	0.5	0.76	0.473684	0.39523	0.528176	0.651431	0.47182	0.274709	0.38081

0.3	0.625	0.6	0.333333	0.35562	0.035379	0.683792	0.96462	0.32048	0.03951
0.3	0.625	1	0.6	0.35562	0.769374	0.683792	0.23063	0.320468	0.71172
0.3	0.75	0.76	0.473684	0.32205	0.301958	0.714226	0.69804	0.369631	0.22761
0.3	0.875	0.6	0.333333	0	0	0	1	0.333333	0.02103
0.3	0.875	1	0.6	0.29377	0.554806	0.743142	0.44519	0.421615	0.51893
0.3	1	0.76	0.473684	0	0	0	1	0.473684	0.02862
0.4	0	0.52	0.230769	0.71552	0.391996	0.43246	0.608	0.100734	0.20475
0.4	0	1	0.6	1	1	0.6	0	0	0.92949
0.4	0.125	0.76	0.473684	0.63616	0.976181	0.48216	0.02382	0.126319	0.70317
0.4	0.25	0.52	0.230769	0.56739	0.204788	0.526416	0.79521	0.154632	0.11436
0.4	0.25	1	0.6	1	1	0.6	0	0	0.92949
0.4	0.375	0.76	0.473684	0.50758	0.756506	0.566296	0.24349	0.185947	0.54339
0.4	0.5	0.52	0.230769	0.45553	0.026851	0.602644	0.97315	0.220508	0.03169
0.4	0.5	0.92	0.565217	0.45555	0.902087	0.602635	0.09791	0.220481	0.77415
0.4	0.625	0.76	0.473684	0.41036	0.569997	0.636121	0.43	0.258363	0.4121
0.4	0.75	0.52	0.230769	0	0	0	1	0.230769	0.01942
0.4	0.75	0.92	0.565217	0.37127	0.722404	0.667301	0.2776	0.29956	0.62064
0.4	0.875	0.68	0.411765	0.33762	0.192323	0.696637	0.80768	0.343931	0.14126
0.4	1	0.52	0.230769	0	0	0	1	0.230769	0.01942
0.4	1	0.92	0.565217	0.30888	0.522559	0.724468	0.47744	0.390918	0.45564
0.5	0	0.68	0.411765	1	1	0.411765	0	0	0.65259
0.5	0.125	0.52	0.230769	0.71885	0.40643	0.424168	0.59357	0.098345	0.21353
0.5	0.125	0.92	0.565217	1	1	0.565217	0	0	0.86171
0.5	0 25	0.68	0.411765	0.64437	0.829491	0.471346	0.17051	0.121914	0.54024
0.5	0.375	0.44	0.090909	0	0	0	1	0.090909	0.01684
0.5	0.375	0.92	0.565217	1	1	0.565217	0	0	0.86171
0.5	0.5	0.68	0.411765	0.52196	0.626877	0.551949	0.37312	0.176244	0.40955
0.5	0.625	0.44	0.090909	0	0	0	1	0.090909	0.01684
0.5	0.625	0.92	0.565217	0.47165	0.942632	0.586992	0.05737	0.207424	0.81148
0.5	0.75	0.68	0.411765	0.42748	0.450636	0.619358	0.54936	0.241479	0.29953
0.5	0.875	0.44	0.090909	0	0	0	1	0.090909	0.01684
0.5	0.875	0.84	0.52381	0.38879	0.661219	0.649527	0.33878	0.27844	0.52497
0.5	1	0.68	0.411765	0.35505	0.260077	0.677904	0.73992	0.318219	0.18432
0.6	0	0.44	0.090909	0.89845	0.137494	0.305576	0.86251	0.056688	0.07459
0.6	0	0.84	0.52381	1	1	0.52381	0	0	0.79359
0.6	0.125	0.68	0.411765	1	1	0.411765	0	0	0.65388
0.6	0.25	0.44	0.090909	0	0	0	1	0.090909	0.01857
0.6	0.25	0.84	0.52381	1	1	0.52381	0	0	0.79359
0.6	0.375	0.6	0.333333	0.65596	0.634599	0.458227	0.3654	0.116428	0.37208
0.6	0.5	0.44	0.090909	0	0	0	1	0.090909	0.01857
0.6	0.5	0.84	0.52381	1	1	0.52381	0	0	0.79359
0.6	0.625	0.6	0.333333	0.5387	0.452728	0.535864	0.54727	0.165791	0.26983
0.6	0.625	1	0.6	1	1	0.6	0	0	0.93354
0.6	0.75	0.84	0.52381	0.48984	0.87764	0.569808	0.12236	0.193876	0.69574
0.6	0.875	0.6	0.333333	0.44651	0.289037	0.601223	0.71096	0.224424	0.18077
0.6	0.875	1	0.6	0.44651	0.996753	0.601223	0.00325	0.224415	0.93047

0.6	1	0.84	0.52381	0.40815	0.713914	0.630536	0.28609	0.257479	0.56781
0.7	0	0.6	0.333333	1	1	0.333333	0	0	0.58539
0.7	0	1	0.6	1	1	0.6	0	0	0.93557
0.7	0.125	0.76	0.473684	1	1	0.473684	0	0	0.72516
0.7	0.25	0.6	0.333333	0.81335	0.935165	0.35145	0.06483	0.072017	0.54745
0.7	0.25	1	0.6	1	1	0.6	0	0	0.93557
0.7	0.375	0.76	0.473684	1	1	0.473684	0	0	0.72516
0.7	0.5	0.6	0.333333	0.67055	0.670567	0.442998	0.32943	0.110108	0.39426
0.7	0.5	1	0.6	1	1	0.6	0	0	0.93557
0.7	0.625	0.76	0.473684	0.61088	0.975521	0.482269	0.02448	0.131578	0.70726
0.7	0.75	0.52	0.230769	0.55761	0.208933	0.518143	0.79107	0.154869	0.12214
0.7	0.75	1	0.6	1	1	0.6	0	0	0.93557
0.7	0.875	0.76	0.473684	0.50999	0.791177	0.551166	0.20882	0.180125	0.57402
0.7	1	0.52	0.230769	0.46736	0.062223	0.581782	0.93778	0.207479	0.05432
0.7	1	1	0.6	1	1	0.6	0	0	0.93557
0.8	0	0.76	0.473684	1	1	0.473684	0	0	0.72664
0.8	0.125	0.52	0.230769	1	1	0.230769	0	0	0.51693
0.8	0.125	0.92	0.565217	1	1	0.565217	0	0	0.86724
0.8	0.25	0.76	0.473684	1	1	0.473684	0	0	0.72664
0.8	0.375	0.52	0.230769	0.82405	0.603305	0.337611	0.3967	0.068282	0.31604
0.8	0.375	0.92	0.565217	1	1	0.565217	0	0	0.86724
0.8	0.5	0.68	0.411765	1	1	0.411765	0	0	0.65647
0.8	0.625	0.52	0.230769	0.68782	0.395466	0.425811	0.60453	0.103179	0.21322
0.8	0.625	0.92	0.565217	1	1	0.565217	0	0	0.86724
0.8	0.75	0.68	0.411765	0.63028	0.847147	0.463925	0.15285	0.122677	0.55649
0.8	0.875	0.52	0.230769	0.57856	0.245133	0.498865	0.75487	0.143709	0.14111
0.8	0.875	0.92	0.565217	1	1	0.565217	0	0	0.86724
0.8	1	0.68	0.411765	0.53199	0.672774	0.53111	0.32723	0.166392	0.44448
0.9	0	0.44	0.090909	1	1	0.090909	0	0	0.44907
0.9	0	0.92	0.565217	1	1	0.565217	0	0	0.86909
0.9	0.125	0.68	0.411765	1	1	0.411765	0	0	0.65777
0.9	0.25	0.44	0.090909	1.00198	0.304681	0.216276	0.69532	0.035975	0.14954
0.9	0.25	0.92	0.565217	1	1	0.565217	0	0	0.86909
0.9	0.375	0.68	0.411765	1	1	0.411765	0	0	0.65777
0.9	0.5	0.44	0.090909	0.8382	0.104902	0.321386	0.8951	0.063898	0.06601
0.9	0.5	0.84	0.52381	1	1	0.52381	0	0	0.79858
0.9	0.625	0.68	0.411765	1	1	0.411765	0	0	0.65777
0.9	0.75	0.44	0.090909	0	0	0	1	0.090909	0.02379
0.9	0.75	0.84	0.52381	1	1	0.52381	0	0	0.79858
0.9	0.875	0.68	0.411765	0.65181	0.902654	0.443929	0.09735	0.113513	0.59412
0.9	1	0.44	0.090909	0	0	0	1	0.090909	0.02379
0.9	1	0.84	0.52381	1	1	0.52381	0	0	0.79858
1	0	0.6	0.333333	1	1	0.333333	0	0	0.58872
1	0.125	0.44	0.090909	1	1	0.090909	0	0	0.44981
1	0.125	0.84	0.52381	1	1	0.52381	0	0	0.80025
1	0.25	0.6	0.333333	1	1	0.333333	0	0	0.58872

1	0.25	1	0.6	1	1	0.6	0	0	0.94168
1	0.375	0.84	0.52381	1	1	0.52381	0	0	0.80025
1	0.5	0.6	0.333333	1	1	0.333333	0	0	0.58872
1	0.5	1	0.6	1	1	0.6	0	0	0.94168
1	0.625	0.84	0.52381	1	1	0.52381	0	0	0.80025
1	0.75	0.6	0.333333	0.78923	0.952228	0.346386	0.04777	0.073149	0.56097
1	0.75	1	0.6	1	1	0.6	0	0	0.94168
1	0.875	0.76	0.473684	1	1	0.473684	0	0	0.72961
1	1	0.6	0.333333	0.67533	0.720192	0.422343	0.27981	0.104232	0.42734
1	1	1	0.6	1	1	0.6	0	0	0.94168

Test data for M-W flash

Filename: fts70k.txt

Number of data points: 70 with no error in K values

T	P	F	z	K	V	y	L	x	q
0.075	0.18	0.66	0.3939	0.38369	0.27414	0.6683	0.72586	0.29031	0.176409
0.075	0.18	0.92	0.5652	0.38369	0.72726	0.6683	0.27274	0.2903	0.616376
0.075	0.355	0.66	0.3939	0.32414	0.06968	0.7188	0.93032	0.36961	0.057213
0.075	0.355	0.92	0.5652	0.32414	0.5602	0.7188	0.4398	0.36959	0.476506
0.075	0.53	0.66	0.3939	0	0	0	1	0.39394	0.017072
0.075	0.53	0.92	0.5652	0.27938	0.35246	0.7651	0.64754	0.45642	0.307009
0.075	0.705	0.66	0.3939	0	0	0	1	0.39394	0.017072
0.075	0.705	0.92	0.5652	0.24603	0.06701	0.8085	0.93299	0.54774	0.07888
0.075	0.75	0.66	0.3939	0	0	0	1	0.39394	0.017072
0.075	0.75	0.92	0.5652	0	0	0	1	0.56522	0.025842
0.215	0.18	0.66	0.3939	0.47075	0.46909	0.5996	0.53091	0.21227	0.296105
0.215	0.18	0.92	0.5652	0.47075	0.91134	0.5996	0.08866	0.21227	0.778366
0.215	0.355	0.66	0.3939	0.39796	0.31846	0.6523	0.68154	0.2732	0.206017
0.215	0.355	0.92	0.5652	0.39796	0.77021	0.6523	0.22979	0.2732	0.65643
0.215	0.53	0.66	0.3939	0.34054	0.14422	0.6996	0.85578	0.34242	0.104235
0.215	0.53	0.92	0.5652	0.34055	0.6237	0.6996	0.3763	0.34241	0.533187
0.215	0.705	0.66	0.3939	0	0	0	1	0.39394	0.021301
0.215	0.705	0.92	0.5652	0.29589	0.4519	0.7431	0.54811	0.41857	0.392277
0.215	0.75	0.66	0.3939	0	0	0	1	0.39394	0.021301
0.215	0.75	0.92	0.5652	0.28617	0.40093	0.7538	0.59907	0.43902	0.351041
0.355	0.18	0.66	0.3939	0.57035	0.64381	0.5267	0.35619	0.15392	0.406364
0.355	0.18	0.92	0.5652	1	1	0.5652	0	0	0.859044
0.355	0.355	0.66	0.3939	0.4858	0.50597	0.5832	0.49404	0.20009	0.32148
0.355	0.355	0.92	0.5652	0.4858	0.95302	0.5832	0.04698	0.20009	0.817693
0.355	0.53	0.66	0.3939	0.41674	0.37076	0.6329	0.62924	0.25313	0.240325

0.355	0.53	0.92	0.5652	0.41675	0.82175	0.6329	0.17825	0.25312	0.703822
0.355	0.705	0.66	0.3939	0.36073	0.22174	0.6776	0.77826	0.31311	0.152928
0.355	0.705	0.92	0.5652	0.36075	0.69163	0.6776	0.30837	0.31308	0.593658
0.355	0.75	0.66	0.3939	0.34819	0.17925	0.6885	0.82075	0.32961	0.128314
0.355	0.75	0.92	0.5652	0.34821	0.65654	0.6885	0.34346	0.32955	0.564402
0.495	0.18	0.66	0.3939	0.6807	0.8389	0.4485	0.1611	0.10981	0.531834
0.495	0.18	0.92	0.5652	1	1	0.5652	0	0	0.861614
0.495	0.355	0.66	0.3939	0.5853	0.68209	0.5099	0.31791	0.14519	0.432819
0.495	0.355	0.92	0.5652	1	1	0.5652	0	0	0.861614
0.495	0.53	0.66	0.3939	0.50589	0.55181	0.5632	0.44819	0.18555	0.352315
0.495	0.53	0.92	0.5652	1	1	0.5652	0	0	0.861614
0.495	0.705	0.66	0.3939	0.43966	0.42882	0.6104	0.57119	0.23141	0.278084
0.495	0.705	0.92	0.5652	0.43967	0.8807	0.6104	0.1193	0.2314	0.757656
0.495	0.75	0.66	0.3939	0.42451	0.39669	0.6218	0.60331	0.24413	0.258974
0.495	0.75	0.92	0.5652	0.42451	0.85021	0.6218	0.14979	0.24412	0.731424
0.635	0.18	0.66	0.3939	1	1	0.3939	0	0	0.636894
0.635	0.18	0.92	0.5652	1	1	0.5652	0	0	0.864192
0.635	0.355	0.66	0.3939	0.69496	0.88673	0.4311	0.11327	0.10338	0.564388
0.635	0.355	0.92	0.5652	1	1	0.5652	0	0	0.864192
0.635	0.53	0.66	0.3939	0.60587	0.73174	0.489	0.26826	0.13453	0.466349
0.635	0.53	0.92	0.5652	1	1	0.5652	0	0	0.864192
0.635	0.705	0.66	0.3939	0.53046	0.60586	0.5399	0.39415	0.16963	0.388201
0.635	0.705	0.92	0.5652	1	1	0.5652	0	0	0.864192
0.635	0.75	0.66	0.3939	0.51298	0.57587	0.552	0.42413	0.17934	0.369836
0.635	0.75	0.92	0.5652	1	1	0.5652	0	0	0.864192
0.775	0.18	0.66	0.3939	1	1	0.3939	0	0	0.638641
0.775	0.18	0.92	0.5652	1	1	0.5652	0	0	0.866778
0.775	0.355	0.66	0.3939	1	1	0.3939	0	0	0.638641
0.775	0.355	0.92	0.5652	1	1	0.5652	0	0	0.866778
0.775	0.53	0.66	0.3939	0.71544	0.95114	0.4093	0.04886	0.09534	0.607412
0.775	0.53	0.92	0.5652	1	1	0.5652	0	0	0.866778
0.775	0.705	0.66	0.3939	0.6313	0.79332	0.4646	0.20668	0.12266	0.507345
0.775	0.705	0.92	0.5652	1	1	0.5652	0	0	0.866778
0.775	0.75	0.66	0.3939	0.61167	0.75889	0.4777	0.24111	0.13017	0.485731
0.775	0.75	0.92	0.5652	1	1	0.5652	0	0	0.866778
0.9	0.18	0.66	0.3939	1	1	0.3939	0	0	0.640204
0.9	0.18	0.92	0.5652	1	1	0.5652	0	0	0.869094
0.9	0.355	0.66	0.3939	1	1	0.3939	0	0	0.640204
0.9	0.355	0.92	0.5652	1	1	0.5652	0	0	0.869094
0.9	0.53	0.66	0.3939	1	1	0.3939	0	0	0.640204
0.9	0.53	0.92	0.5652	1	1	0.5652	0	0	0.869094
0.9	0.705	0.66	0.3939	1	1	0.3939	0	0	0.640204
0.9	0.705	0.92	0.5652	1	1	0.5652	0	0	0.869094
0.9	0.75	0.66	0.3939	0.70753	0.9587	0.4068	0.04131	0.09582	0.613885
0.9	0.75	0.92	0.5652	1	1	0.5652	0	0	0.869094

Training file for M-W system in random order for MLP

Number of data points: 150

T	P	F	z	K	V	y	L	x	q
1	0.75	0.44	0.0909	0.78923	0.065	0.346	0.935	0.073	0.0515
0.2	0.75	0.68	0.4118	0.28891	0	0	1	0.412	0.0217
0.9	0.25	0.76	0.4737	0.83041	1	0.474	0	0	0.7281
0.7	0.625	0.84	0.5238	0.58305	1	0.524	0	0	0.7952
0.5	0	0.44	0.0909	0.80438	0.048	0.371	0.952	0.077	0.0361
0.8	1	1	0.6	0.48572	1	0.6	0	0	0.9376
0	0.25	0.52	0.2308	0.3778	0	0	1	0.231	0.0107
0.1	0.75	1	0.6	0.24633	0.205	0.808	0.795	0.546	0.2063
0.7	0.75	1	0.6	0.50114	1	0.6	0	0	0.9356
0.7	0.875	0.44	0.0909	0.59641	0	0	1	0.091	0.0203
0.4	0.875	0.52	0.2308	0.38495	0	0	1	0.231	0.0194
0.6	0.375	0.84	0.5238	0.61096	1	0.524	0	0	0.7936
0.8	0.25	0.68	0.4118	0.8269	1	0.412	0	0	0.6565
1	1	0.52	0.2308	0.67533	0.398	0.422	0.602	0.104	0.2178
0.7	0.625	0.92	0.5652	0.55346	1	0.565	0	0	0.8654
0.3	0	0.84	0.5238	0.60833	1	0.524	0	0	0.7886
0.8	0.25	0.84	0.5238	0.73728	1	0.524	0	0	0.7969
0.1	0.875	1	0.6	0.22908	0	0	1	0.6	0.0298
0	0	0.68	0.4118	0.41541	0.393	0.647	0.607	0.26	0.2521
0.5	0.5	0.44	0.0909	0.60794	0	0	1	0.091	0.0168
0	0.875	1	0.6	0.21245	0	0	1	0.6	0.0248
0.1	0.875	0.52	0.2308	0.30999	0	0	1	0.231	0.0129
0.9	1	0.68	0.4118	0.60142	0.808	0.478	0.192	0.132	0.5328
0.8	0.625	0.76	0.4737	0.65691	1	0.474	0	0	0.7266
0.8	0.125	1	0.6	0.71107	1	0.6	0	0	0.9376
0.4	0.875	0.52	0.2308	0.38495	0	0	1	0.231	0.0194
0	0	0.84	0.5238	0.41541	0.682	0.647	0.318	0.26	0.5286
0.6	0.375	1	0.6	0.5515	1	0.6	0	0	0.9335
0.5	0.75	0.6	0.3333	0.42748	0.243	0.619	0.757	0.241	0.1539
0	0.75	0.76	0.4737	0.23806	0	0	1	0.474	0.0178
0.5	1	0.92	0.5652	0.35506	0.687	0.678	0.313	0.318	0.5935
0.4	0.25	0.76	0.4737	0.56738	0.858	0.526	0.142	0.155	0.6167
0.2	0.375	0.6	0.3333	0.38218	0.115	0.665	0.885	0.29	0.0789
0.4	0.25	0.52	0.2308	0.56739	0.205	0.526	0.795	0.155	0.1144
0.6	1	0.44	0.0909	0.53237	0	0	1	0.091	0.0186
0.7	0.25	0.52	0.2308	0.81335	0.568	0.351	0.432	0.072	0.297
0.7	0.875	0.52	0.2308	0.50998	0.136	0.551	0.864	0.18	0.0884
0.5	1	0.6	0.3333	0.35505	0.042	0.678	0.958	0.318	0.0482
0.3	0	0.84	0.5238	0.60833	1	0.524	0	0	0.7886
0.8	1	0.92	0.5652	0.50947	1	0.565	0	0	0.8672

0.4	0	0.92	0.5652	0.61727	1	0.565	0	0	0.8599
0.7	0.5	1	0.6	0.55577	1	0.6	0	0	0.9356
0.3	0.375	0.84	0.5238	0.44166	0.758	0.617	0.242	0.233	0.5953
0	0	1	0.6	0.41541	0.878	0.647	0.122	0.26	0.8054
0.1	0.125	0.6	0.3333	0.42132	0.208	0.639	0.792	0.253	0.1262
0.5	0.75	0.76	0.4737	0.42748	0.614	0.619	0.386	0.241	0.4453
0.8	0.5	0.92	0.5652	0.62182	1	0.565	0	0	0.8672
0.2	0.375	0.92	0.5652	0.3822	0.734	0.665	0.266	0.29	0.6249
0	0.625	0.68	0.4118	0.26198	0	0	1	0.412	0.0154
0	0.75	1	0.6	0.22307	0	0	1	0.6	0.0248
0.1	0.75	0.92	0.5652	0.24633	0.072	0.808	0.928	0.546	0.084
0.7	1	0.84	0.5238	0.46737	0.845	0.582	0.155	0.207	0.6721
0.3	0.25	0.68	0.4118	0.49579	0.565	0.579	0.435	0.195	0.3666
0	0.625	0.44	0.0909	0.4045	0	0	1	0.091	0.0083
0.5	0.25	0.6	0.3333	0.64437	0.605	0.471	0.395	0.122	0.3535
0.5	0	0.6	0.3333	0.80439	0.872	0.371	0.128	0.077	0.508
0.5	0.75	0.68	0.4118	0.42748	0.451	0.619	0.549	0.241	0.2995
0.4	1	0.52	0.2308	0.36745	0	0	1	0.231	0.0194
0.1	0.625	1	0.6	0.26874	0.4	0.777	0.6	0.482	0.3752
0.3	0.125	0.6	0.3333	0.55872	0.459	0.537	0.541	0.16	0.2677
0.9	0.125	0.76	0.4737	0.88577	1	0.474	0	0	0.7281
0.7	0.625	0.92	0.5652	0.55346	1	0.565	0	0	0.8654
0.6	0.375	0.52	0.2308	0.65595	0.335	0.458	0.665	0.116	0.1801
0.7	0.5	0.68	0.4118	0.67055	0.906	0.443	0.094	0.11	0.5935
0.3	0.625	0.92	0.5652	0.35562	0.674	0.684	0.326	0.32	0.5771
0	0.625	0.84	0.5238	0.2433	0	0	1	0.524	0.0201
1	0.875	0.84	0.5238	0.63719	1	0.524	0	0	0.8003
0.5	1	0.84	0.5238	0.35506	0.572	0.678	0.428	0.318	0.457
0	0	0.52	0.2308	0.43177	0	0	1	0.231	0.0107
0.9	0.375	0.92	0.5652	0.70128	1	0.565	0	0	0.8691
0	0.25	1	0.6	0.32036	0.642	0.725	0.358	0.377	0.587
0.4	0.125	0.68	0.4118	0.63617	0.802	0.482	0.198	0.126	0.5208
0.5	0.375	0.68	0.4118	0.57919	0.722	0.514	0.278	0.148	0.4701
0.6	0.375	0.44	0.0909	0.68895	0	0	1	0.091	0.0186
0.8	0.125	0.52	0.2308	0.99571	1	0.231	0	0	0.5169
0.7	0.5	0.6	0.3333	0.67055	0.671	0.443	0.329	0.11	0.3943
0.5	1	0.44	0.0909	0.49741	0	0	1	0.091	0.0168
0	0.375	0.76	0.4737	0.2861	0.099	0.759	0.901	0.442	0.083
0.1	0.375	0.52	0.2308	0.38293	0	0	1	0.231	0.0129
1	0.75	0.6	0.3333	0.78923	0.952	0.346	0.048	0.073	0.561
0.2	0	0.84	0.5238	0.55363	0.944	0.545	0.056	0.164	0.7414
0.3	0.875	0.92	0.5652	0.29377	0.447	0.743	0.553	0.422	0.3908
0.6	0.75	0.92	0.5652	0.48984	0.988	0.57	0.012	0.194	0.8529
0.6	0.75	0.52	0.2308	0.48983	0.098	0.57	0.902	0.194	0.0686
0.1	0.375	0.68	0.4118	0.33012	0.147	0.713	0.853	0.36	0.1061
0.6	0	0.84	0.5238	0.7412	1	0.524	0	0	0.7936

1	0.5	0.68	0.4118	0.83307	1	0.412	0	0	0.6591
0.3	1	0.6	0.3333	0.30309	0	0	1	0.333	0.021
0.9	0.125	1	0.6	0.75756	1	0.6	0	0	0.9396
1	0.75	0.68	0.4118	0.74976	1	0.412	0	0	0.6591
0.2	0.875	0.84	0.5238	0.2574	0.045	0.79	0.955	0.511	0.0605
0.2	0.75	1	0.6	0.28041	0.479	0.761	0.521	0.452	0.4486
0.4	0.75	0.44	0.0909	0.51076	0	0	1	0.091	0.0151
0.8	1	0.68	0.4118	0.53199	0.673	0.531	0.327	0.166	0.4445
0.9	0.875	1	0.6	0.54111	1	0.6	0	0	0.9396
0.2	0.75	0.52	0.2308	0.35019	0	0	1	0.231	0.015
0.8	0.375	0.6	0.3333	0.82405	0.984	0.338	0.016	0.068	0.5772
0.7	0.625	0.92	0.5652	0.55346	1	0.565	0	0	0.8654
0.4	1	0.84	0.5238	0.30887	0.398	0.724	0.602	0.391	0.3252
0.2	1	0.44	0.0909	0.40351	0	0	1	0.091	0.0117
0.7	0.25	1	0.6	0.62524	1	0.6	0	0	0.9356
0.1	0.25	0.76	0.4737	0.37139	0.455	0.677	0.545	0.304	0.3253
0.2	0.25	0.84	0.5238	0.43018	0.727	0.629	0.273	0.244	0.5689
0.1	0.25	0.6	0.3333	0.37139	0.079	0.677	0.921	0.304	0.0572
0.6	0	0.52	0.2308	0.89846	0.699	0.306	0.301	0.057	0.3618
0	0	0.76	0.4737	0.41541	0.552	0.647	0.448	0.26	0.3903
0.1	1	0.68	0.4118	0.24389	0	0	1	0.412	0.0185
0.6	0.625	0.6	0.3333	0.5387	0.453	0.536	0.547	0.166	0.2698
0.4	0.75	0.92	0.5652	0.37127	0.722	0.667	0.278	0.3	0.6206
0	1	0.52	0.2308	0.27476	0	0	1	0.231	0.0107
0.7	0.625	0.44	0.0909	0.65919	0	0	1	0.091	0.0203
0.4	0.125	0.52	0.2308	0.63619	0.294	0.482	0.706	0.126	0.1568
0.9	0.125	0.92	0.5652	0.79478	1	0.565	0	0	0.8691
0.1	0.375	0.52	0.2308	0.38293	0	0	1	0.231	0.0129
1	0.75	0.6	0.3333	0.78923	0.952	0.346	0.048	0.073	0.561
0	0.25	1	0.6	0.32036	0.642	0.725	0.358	0.377	0.587
0	0.875	1	0.6	0.21245	0	0	1	0.6	0.0248
0.5	0.375	1	0.6	0.51748	1	0.6	0	0	0.9315
0.8	0.5	0.52	0.2308	0.75205	0.487	0.384	0.513	0.085	0.2584
1	0.25	0.68	0.4118	0.9371	1	0.412	0	0	0.6591
0.2	0.75	0.92	0.5652	0.28041	0.366	0.761	0.634	0.452	0.3226
0.3	1	0.76	0.4737	0.27041	0	0	1	0.474	0.0286
0.8	0.5	0.52	0.2308	0.75205	0.487	0.384	0.513	0.085	0.2584
0.5	0.5	0.68	0.4118	0.52196	0.627	0.552	0.373	0.176	0.4096
0.8	0.25	0.52	0.2308	0.90519	0.764	0.286	0.236	0.053	0.3968
0.7	0.375	0.76	0.4737	0.6889	1	0.474	0	0	0.7252
0.7	0.5	0.68	0.4118	0.67055	0.906	0.443	0.094	0.11	0.5935
0.8	0.25	1	0.6	0.66663	1	0.6	0	0	0.9376
0	0.375	0.76	0.4737	0.2861	0.099	0.759	0.901	0.442	0.083
0.2	0.5	0.92	0.5652	0.34189	0.627	0.699	0.373	0.341	0.5353
0.2	0.375	1	0.6	0.38221	0.826	0.665	0.174	0.29	0.7615
0.6	0.125	0.84	0.5238	0.69178	1	0.524	0	0	0.7936

0.7	0.125	0.76	0.4737	0.78043	1	0.474	0	0	0.7252
0.6	0.875	0.44	0.0909	0.55772	0	0	1	0.091	0.0186
0.4	0	0.92	0.5652	0.61727	1	0.565	0	0	0.8599
0.2	0.5	0.6	0.3333	0.34443	0	0	1	0.333	0.0184
0.2	0.625	0.52	0.2308	0.36862	0	0	1	0.231	0.015
0.7	0.125	0.84	0.5238	0.73788	1	0.524	0	0	0.7952
0.1	0.875	0.68	0.4118	0.2555	0	0	1	0.412	0.0185
0.8	0.125	0.84	0.5238	0.78643	1	0.524	0	0	0.7969
0.1	0.75	0.92	0.5652	0.24633	0.072	0.808	0.928	0.546	0.084
0	0.5	0.84	0.5238	0.25868	0.047	0.793	0.953	0.511	0.0541
0.7	0.25	0.76	0.4737	0.73195	1	0.474	0	0	0.7252
0.1	0.375	0.6	0.3333	0.33875	0	0	1	0.333	0.0157
1	0.375	1	0.6	0.7116	1	0.6	0	0	0.9417
0.5	0.25	1	0.6	0.54982	1	0.6	0	0	0.9315
0.6	0.875	0.68	0.4118	0.44651	0.497	0.601	0.503	0.224	0.3305
1	0	0.68	0.4118	1.07097	1	0.412	0	0	0.6591
0	0.125	0.92	0.5652	0.36295	0.671	0.687	0.329	0.316	0.5671
0.6	0.5	0.92	0.5652	0.54765	1	0.565	0	0	0.8635

The following training and testing data files are in the enclosed diskette.

Training file for M-W system in two-phase region only

File name: ftrnz1 (100 data points)

Order of variables is the same as the previous training files for the M-W system.

Testing file for M-W system in two-phase region only

File name: ftstnz2 (70 data points).

Order of variables is the same as the previous training files for the M-W system.

Training file for B-T system

File name: fbt2300 (300 data points).

Order of variables: T, P, F, z, V, y, L, x, and q

Testing file for B-T system

File name: fbs100 (100 data points)

Order of variables: T, P, F, z, V, y, L, x, and q

C.1 TRAINING AND TESTING FILES FOR THE CRUDE DISTILLATION TOWER

1. Top and heavy naphta section

Training file: HT300 (300 data points).

Testing file: HS100 (100 data points).

Order of variables in both files: Bintolt, Htfeed, Hndraw, Kerodraw, Qreb, Ttop, Ovhd, RR, Qcond, PAT, Ttoph, Tboth, PAH, IBPH, FBPH, RHOH

2. Kerosene section

Training file: Ktc20r (200 data points).

Testing file: Ksc10r (100 data points).

Order of variables in both files: Bintolt, Htfeed, Hndraw, Kerodraw, Diesdraw, SSK, TtopK, TbotK, FPKero, IBPK, FBPK.

3. Diesel section

Training file: Dtc620 (200 data points).

Testing file: Dsc610 (100 data points).

Order of variables in both files: Bintolt, Htfeed, Kerodraw, Diesdraw, AGOdraw, SSD, TtopD, TbotD, IBPD, FBPD, PourD, PAD.

4. AGO section

Training file: Agt150 (150 data points).

Testing file: Ags100 (100 data points).

Order of variables in both files: Bintolt, Htfeed, Diesdraw, AGOdraw, SSA, TtopA, TbotA, IBPA, FBPA, PourA, PAA.

5. LSWR section

Training file: Lswt150 (150 data points).

Testing file: Lsws70 (70 data points).

Order of variables in both files: Bintolt, Htfeed, AGOdraw, SSM, Tbot, PourL.

APPENDIX D

MATLAB PROGRAMS

D.1 Sample Program for Standard RBFN.

```
% ********** Function approximation using RBFN
clear; clf;
N=150; NR=20;

% define an input vectors and a target vector y.
f=fopen('./flth315','r');

for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x, y will be stored as a row in array A
end
fclose(f);

p = (A(:,1:4))';
t = (A(1:N,6:10))';

x1= A(1:N,1); x2= A(1:N,2); x3= A(1:N,3); x4= A(:,4); x5= A(:,5);
y1= A(1:N,6); y2= A(1:N,7); y3= A(1:N,8); y4= A(:,9); y5= A(:,10);

% Identification using an RBFN
% train the RBFN

df=100;   % Frequency of progress displays (just once).
me=5000; % Maximum number of epochs/neurons (as many as it takes).
sse=0.01; % Sum-squared error goal.
sc=.15;     % Spread constant for radial basis functions.
tp = [df me sse sc];

% Training begin...

flops(0), tic

mysrb
```

```
rbfn_flops = flops
rbfn_time = toc

%      ..end of training.


% ***plot training data
out = simurb(p,w1,b1,w2,b2);

subplot(2,1,1),plot3(x1,x2,out(1,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y1'); view(-30,30);
title('Training data (y1)');
plot3(x1,x2,y1,'go');

tmp=out(1,:);
for i=1:N,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y1(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end

% ***test network
f=fopen('./fts70k','r');
for i = 1:70
        aa=fscanf(f,'%f',[1 10]);
        AA=[AA;aa];        %x1, x2, y will be stored as a row in array
A
end
fclose(f);

x1=[]; x2=[]; x3=[]; x4=[]; x5=[];
y1=[]; y2=[]; y3=[]; y4=[]; y5=[];
x1=AA(:,1)'; x2=AA(:,2)'; x3=AA(:,3)'; x4=AA(:,4)'; x5=AA(:,5)';
y1=AA(:,6)'; y2=AA(:,7)'; y3=AA(:,8)'; y4=AA(:,9)'; y5=AA(:,10)'

%%%%%*****  test with trained/untrained data ****%%%%%

g=[x1; x2; x3; x4];
out = simurb(g,w1,b1,w2,b2);

yy = [y1; y2; y3; y4; y5];
sse = sumsqr(yy - out)
RMS = sqrt(sse/NR)
sse1 = sumsqr(y1 - out(1,:))
sse2 = sumsqr(y2 - out(2,:))
sse3 = sumsqr(y3 - out(3,:))
sse4 = sumsqr(y4 - out(4,:))
sse5 = sumsqr(y5 - out(5,:))
rms1 = sqrt(sse1/NR)
rms2 = sqrt(sse2/NR)
rms3 = sqrt(sse3/NR)
```

```
rms4 = sqrt(sse4/NR)
rms5 = sqrt(sse5/NR)
epoch

subplot(2,1,2),plot3(x1,x2,out(1,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y1'); view(-30,30);
title('Test data (y1)');
plot3(x1,x2,y1,'go');

tmp=out(1,:);
for i=1:NR,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y1(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end
```

D.2 Sample Program for MLP

```
clear;clc;

N=300;        % Number of Training samples
df=500;       % Frequency of progress displays (just once).
me=10000;     % Maximum number of epochs/neurons (as many as it
              % takes).
rms=0.1;      % Maximum acceptable root mean squar error
lr=0.01;      % Learning rate.
tp = [df me rms lr];

FIRST=30;     % Number of Neurons in the first hidden layer
SECOND=10;

% define an input vectors and a target vector y.
f=fopen('./flth815','r');

A=[];
for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x1, x2, y will be stored as a row in array A
end
fclose(f);

p = (A(:,1:4))';
t = (A(:,6))';
```

```
x1= A(1:N,1); x2= A(1:N,2); x3= A(1:N,3); x4= A(1:N,4); %x5= A(:,5);
y1= A(1:N,6); %y2= A(1:N,7); y3= A(1:N,8); y4= A(:,9); y5= A(1:N,10);

%    ================================
%      Identification using an BP Network
%    ================================


[w1,b1,w2,b2,w3,b3]=initff(p,FIRST,'logsig',SECOND,'logsig',t,'purelin
');

%       Training begin...

        flops(0), tic


    [w1,b1,w2,b2,w3,b3,bp_ep]=mytrbpx(w1,b1,'logsig',w2,b2,'logsig',...
    w3,b3,'purelin',p,t,tp);

        bp_flops = flops;
        bp_time = toc;

%       ..end of training.

        bp_flops
        bp_time


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Plot Training Data    %%%%%

clf
out = simuff(p,w1,b1,'logsig',w2,b2,'logsig',w3,b3,'purelin');

subplot(2,2,1),plot3(x1,x2,out(1,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y1'); view(-30,30);
title('Training data (y1)');
plot3(x1,x2,y1,'go');

tmp=out(1,:);
for i=1:N,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y1(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end

subplot(2,2,2); plot3(x1,x2,out(2,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y2'); view(-30,30);
title('Training data (y2)');
plot3(x1,x2,y2,'go');
```

```
tmp=out(2,:);
for i=1:N,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y2(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Plot Test Data %%%%
A = []; a = [];
f=fopen('./fts70k','r');
NR = 100;
x1= []; x2= []; x3= []; x4= []; x5= [];
y1= []; y2= []; y3= []; y4= []; y5= [];


for i = 1:NR
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x1, x2, y will be stored as a row in array A
end
fclose(f);

x1= A(:,1); x2= A(:,2); x3= A(:,3); x4= A(:,4); %x5= A(:,5);
y1= A(:,6); %y2= A(:,7); y3= A(:,8); y4= A(:,9); y5= A(:,10);
g=[x1 x2 x3 x4];

out = simuff(g',w1,b1,'logsig',w2,b2,'logsig',w3,b3,'purelin');
%yy = [y1 y2 y3 y4 y5]';
%sse = sumsqr(yy - out)
%rms = sqrt(sse/(5*NR))
sse1 = sumsqr(y1' - out(1,:))
%sse2 = sumsqr(y2' - out(2,:))
%sse3 = sumsqr(y3' - out(3,:))
%sse4 = sumsqr(y4' - out(4,:))
%sse5 = sumsqr(y5' - out(5,:))
%sse6 = sumsqr(y6 - out(6,:))
rms1 = sqrt(sse1/NR)
%rms2 = sqrt(sse2/NR)
%rms3 = sqrt(sse3/NR)
%rms4 = sqrt(sse4/NR)
%rms5 = sqrt(sse5/NR)
%rms6 = sqrt(sse6/NR)



subplot(2,2,3); plot3(x1,x2,out(1,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y2'); view(-30,30);
title('Test data (y1)');
```

```
plot3(x1,x2,y1,'go');

tmp=out(1,:);
for i=1:NR,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y1(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end

subplot(2,2,4); plot3(x1,x2,out(2,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y2'); view(-30,30);
title('Test data (y2)');
plot3(x1,x2,y2,'go');

tmp=out(2,:);
for i=1:NR,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y2(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end
```

D.3 Sample Program for Hybrid RBF-FPM-RBF

```
% ========== Function approximation using RBFN
clear; clf;
N=150; NR=70; slack=0.02;

% define an input vectors and a target vector y.
f=fopen('./flth150.txt','r');

for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x1, x2, y will be stored as a row in array A
end
fclose(f);

p = (A(:,1:5))';
t = ([A(1:N,6) A(:,9)])';

x1= A(1:N,1); x2= A(1:N,2); x3= A(1:N,3); x4= A(:,4); x5= A(:,5);
y1= A(1:N,6); y2= A(1:N,7); y3= A(1:N,9); %y4= A(:,8); %y5= A(:,9);
```

```
% Identification using an RBFN
% train the RBFN

df=100;   % Frequency of progress displays (just once).
me=5000; % Maximum number of epochs/neurons (as many as it takes).
sse=0.1; % Sum-squared error goal.
sc=.3;     % Spread constant for radial basis functions.
tp = [df me sse sc];

% Training begin...

flops(0), tic

[w11,b11,w12,b12,k1,tr1] = mysrb(p,t,tp);


t      ..end of training.


out = simurb(p,w11,b11,w12,b12);

for i = 1:N
    if out(1,i) > 1
   out(1,i) = 1;
    end
    if out(1,i) < slack
   out(1,i) = 0;
    end
    if out(2,i) > 1
   out(2,i) = 1;
    end
    if out(2,i) < slack
   out(2,i) = 0;
    end
end

%%%%%%%%%% using overall balance  %%%%%%%%%%

F = A(:,3) * 250;
V = out(1,:)' .* F;
L = F - V;

z = A(:,4);
x = out(2,:)';
K = A(:,5) .* 6;
for i=1:N
    num(i) = (F(i)*z(i) - L(i)*x(i));
    num1(i) = K(i)*x(i);
    if (x(i) == 0)
   y(i) = z(i);
   yf(i) = z(i);
    elseif (V(i) == 0)
```

```
   y(i) = 0;
   yf(i) = 0;
    elseif (num1(i) < 0.05)
   y(i) = 0;
   yf(i) = 0;
    elseif ((num1(i) > 1) & (num(i) < 10))
   y(i) = 0;
   yf(i) = 0;
    elseif (num1(i) >1)
   y(i) = 1;
   yf(i) = 1;
    else
   y(i) = num1(i);
   yf(i) = num(i)/V(i);
    end
end

%*****************************************************************
p = ([A(1:N,1:4) y' yf']);
t = (A(1:N,7))';

df=100;    % Frequency of progress displays (just once).
me=5000; % Maximum number of epochs/neurons (as many as it takes).
sse=0.05; % Sum-squared error goal.
sc=.25;      % Spread constant for radial basis functions.
tp = [df me sse sc];

% Training begin...

%flops(0), tic

[w21,b21,w22,b22,k2,tr2] = mysrb(p,t,tp);

rbfn_flops = flops
rbfn_time = toc

out1 = simurb(p,w21,b21,w22,b22);

for i = 1:N
    if out1(i) > 1
   out1(i) = 1;
    end
    if out1(i) < slack
   out1(i) = 0;
    end
end

subplot(2,2,1),plot3(x1,x2,out(1,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y1'); view(-30,30);
title('Training data (y1)');
plot3(x1,x2,y1,'go');
```

```
tmp=out(1,:);
for i=1:N,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y1(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end

subplot(2,2,2); plot3(x1,x2,out1,'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y2');
view(-30,30);
title('Training data (y2)');
plot3(x1,x2,y2,'go');

tmp=out1;
for i=1:N,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y2(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end


% ===test network
f=fopen('./fts70k.txt','r');
for i = 1:70
        aa=fscanf(f,'%f',[1 10]);
        AA=[AA;aa];        %x1, x2. y will be stored as a row in array
A
end
fclose(f);

x1=[]; x2=[]; x3=[]; x4=[]; x5=[]; num=[]; num1=[]; yf=[];
y1=[]; y2=[]; %y3=[]; %y4=[]; y5=[];
F=[]; V=[]; L=[]; y=[]; x=[]; z=[];
x1=AA(:,1)'; x2=AA(:,2)'; x3=AA(:,3)'; x4=AA(:,4)'; x5=AA(:,5)';
y1=AA(:,6)'; y2=AA(:,7)'; y3=AA(:,9)'; %y4=AA(:,8)'; %y5=AA(:,9)'

%%%%%=====  test with trained/untrained data  ====%%%%%

g=[x1; x2; x3; x4; x5];
out = simurb(g,w11,b11,w12,b12);

for i = 1:NR
    if out(1,i) > 1
   out(1,i) = 1;
    elseif out(1,i) < slack
   out(1,i) = 0;
    end
    if out(2,i) > 1
   out(2,i) = 1;
    elseif out(2,i) < slack
```

```
   out(2,i) = 0;
    end
end

F = AA(:,3) * 250;
V = out(1,:)' .* F;
L = F - V;

z = AA(:,4);
x = out(2,:)';
K = AA(:,5) .* 6;

for i=1:NR
    num(i) = (F(i)*z(i) - L(i)*x(i));
    num1(i) = K(i)*x(i);
    if (x(i) == 0)
   y(i) = z(i);
   yf(i) = z(i);
    elseif (V(i) == 0)
   y(i) = 0;
   yf(i) = 0;
    elseif (num1(i) < 0.05)
   y(i) = 0;
   yf(i) = 0;
    elseif ((num1(i) > 1) & (num(i) < 10))
   y(i) = 0;
   yf(i) = 0;
    elseif (num1(i) > 1)
   y(i) = 1;
   yf(i) = 1;
    else
   y(i) = num1(i);
   yf(i) = num(i)/V(i);
    end
end

g=[x1; x2; x3; x4; y; yf];
out1 = simurb(g,w21,b21,w22,b22);

yy = [y1; y3; y2];
sse1 = sqrt((sumsqr(y2 - out1))/NR)
outs = [out; out1];
sse2 = sqrt((sumsqr(yy - outs))/NR)
epoch1 = k1
epoch2 = k2

N=NR;
subplot(2,2,3),plot3(x1,x2,out(1,:),'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y1'); view(-30,30);
title('Test data (y1)');
plot3(x1,x2,y1,'go');
```

```
tmp=out(1,:);
for i=1:N,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y1(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end

subplot(2,2,4); plot3(x1,x2,out1,'r+');
hold on; grid; xlabel('x1'); ylabel('x2'); zlabel('y2');
view(-30,30);
title('Test data (y2)');
plot3(x1,x2,y2,'go');

tmp=out1;
for i=1:N,
        a=x1(i); X=[a;a];
        a=x2(i); Y=[a;a];
        a=tmp(i); b=y2(i); Z=[a;b];
        plot3(X,Y,Z,'y-');
end
```

D.4 Sample Program for Linear-nonlinear HSNN

```
%Linear-nonlinear HSNN
clear all
FIRST=6; % Number of Neurons in the first hidden layer
SECOND=2;    % Number of Neurons in the second hidden layer
N = 150; % Total number of training data set
miu = 0.05; % Learning rate
nepoch =50000;
NT=70;
ssse = 0.08;
RMS = 1;

% define an input vectors and a target vector y.
f=fopen('./Fith315','r');

A=[];
for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x1, x2, y will be stored as a row in array A
end
fclose(f);

p = ([A(:,2) A(:,3:4)])'; % master input
```

```
ps = (A(:,1))';              % slave input
t = (A(:,7))';               % output variable
f1 = 'logsig';
s1 = FIRST;
f2 = 'purelin';
s2 = SECOND;

%Initializing weights and biases
%[V1M,C1M] = feval(feval(f1,'init'),s1,p);
%[row,col]=size(p);
%[V1M,C1M] = rands(s1,row);
%x = ones(s1,1) * feval(f1,'output');
%[S,Q] = size(s2);
%if max(S,Q) > 1, s2 = S; end
%[V2M,C2M] = feval(feval(f2,'init'),s2,x);

% Fixed initial wts and biases to obtain more consistent result
C1M = [-0.85723;-0.6672;0.1388;-0.3251;-0.49891;-0.3237];%./100;
C1M = zeros(6,1);
V1M = [0.55175  0.77052 -0.61304;-0.19768 -0.47102 0.58185;...
-0.50839 0.71472 -0.77083;-0.61587 -0.89650 0.97798;...
-0.10135 -0.21388 -0.52850;-0.64335 0.11409 -0.53792];%./100;
V2M=[0.46043 0.59520 0.28319 0.13708 -0.48899 0.24030;...
0.67126 -0.60252 -0.78367 0.82261 0.57068 0.3445];%./100;
C2M = [-0.69359;-0.14783];%./100;
C2M = zeros(2,1);

% PRESENTATION PHASE
VSSE = [];
Vda = [];
tic;
k = 1;
while RMS > ssse
Verr = [];
for j = 1:N
a1 = feval(f1,V1M*p(:,j),C1M);   % output of first hidden layer
%a2 = feval(f2,V2M*a1,C2M);   % output of master network
a2 = V2M*a1 + C2M;
V1S = a2(1,:); C1S = a2(2,:);
z = (V1S .* ps(:,j)) + C1S;
error = (z-t(:,j));
Verr = [Verr; error];

% Update V's and C's
da1 = a1 .* (1-a1);     % deriv of a1
b1 = 2 .* error .* ps(:,j);
sV2M = sum(V2M);     % summing up V2M column-wise
dint = sV2M.*da1';

C1M1 = C1M - miu.*((2 .* error)*dint)';
V1M1 = V1M - miu.*((2 .* error)*dint)'*p(:,j)';
C1M2 = C1M - miu.*(b1*dint)';
```

```
V1M2 = V1M - miu.*(b1*dint)'*p(:,j)';
C1M = (C1M1+C1M2)./2;
V1M = (V1M1+V1M2)./2;
C2M(1,1) = C2M(1,1) - miu.*(b1);
C2M(2,1) = C2M(2,1) - miu.*(2.*error);
V2M(1,:) = V2M(1,:) - miu.*(b1*a1)';
V2M(2,:) = V2M(2,:) - miu.*(2.*error*a1)';

end

SSE = sumsqr(Verr);
RMS1 = RMS;
RMS = sqrt(SSE/N)
VSSE = [VSSE; RMS];

k = k+1;
if RMS > ssse
   if RMS > RMS1
      loop = k;
      k=nepoch+1;
   end
else
   loop1 = k;
   k=nepoch+1;
   RMStr = RMS
end
end

hstime = toc;

%%%%%%%  testing  %%%%%%%%%%%%%
f=fopen('./fts70k','r');
p=[]; ps=[]; t=[];

A=[];
for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x1, x2, y will be stored as a row in array A
end
fclose(f);

p = ([A(:,2) A(:,3:4)])';
ps = (A(:,1))';
t = (A(:,7))';

Verr = [];
Z=[];
for j = 1:NT
    a1 = feval(f1,V1M*p(:,j),C1M);  % output of first hidden layer
    %a2 = feval(f2,V2M*a1,C2M);     % output of master network
    a2 = V2M*a1 + C2M;
    V1S = a2(1,:); C1S = a2(2,:);
```

```
    z = (V1S .* ps(:,j)) + C1S;
    error = (z-t(:,j));
    Z=[Z z];
    Verr = [Verr; error];
end

SSE = sumsqr(Verr);
MSE = SSE/NT;
RMS = sqrt(MSE)
```

D.5 Sample Program for Nonlinear-nonlinear HSNN

```
clear all
FIRST=20;    % Number of Neurons in the first hidden layer
SECOND=10;   % Number of Neurons in the second hidden layer
N = 150; % Total number of training data set
miu = 0.05; % Learning rate
nepoch =60000;
NT=70;
RMS = 1;
ssse = 0.08;

% define an input vectors and a target vector y.
f=fopen('./Flth815','r');

A=[];
for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x1, x2, y will be stored as a row in array A
end
fclose(f);

p = ([A(:,2) A(:,3:4)])';
ps = (A(:,1))';
t = (A(:,7))';
f1 = 'logsig';
s1 = FIRST;
f2 = 'purelin';
s2 = SECOND;

%Innitialising weights and biases
[V1M,C1M] = feval(feval(f1,'init'),s1,p);
[row,col]=size(p);
[V1M,C1M] = rands(s1,row);
x = ones(s1,1) * feval(f1,'output');
[S,Q] = size(s2);
if max(S,Q) > 1, s2 = S; end
```

```
[V2M,C2M] = feval(feval(f2,'init'),s2,x);


% PRESENTATION PHASE
VSSE = [];
Vda = [];
k = 1;
tic;
while RMS > ssse
Verr = [];
for j = 1:N
a1 = feval(f1,V1M*p(:,j),C1M);   % output of first hidden layer in
master network
a2 = feval(f2,V2M*a1,C2M); % output of master network
V1S = a2(1:3); C1S = a2(4:6); V2S = a2(7:9); C2S = a2(10);
g = feval(f1,V1S*ps(:,j),C1S);   % output of first hidden layer in
slave network
z = (V2S' * g) + C2S;
error = (z-t(:,j));
Verr = [Verr; error];
a111 = [a1 a1 a1];

% Update V's and C's
da1 = a1 .* (1-a1);      % deriv of a1
%da111 = [da1 da1 da1];
dg = g .* (1-g);         % deriv of g
b1 = 2 .* error;
sV2M13 = sum(V2M(1:3,:))'.*da1;      % summing up V2M column-wise
sV2M46 = sum(V2M(4:6,:))'.*da1;
sV2M79 = sum(V2M(7:9,:))'.*da1;
sV2M10 = sum(V2M(10,:));
da113 = [sV2M13 sV2M13 sV2M13];
da146 = [sV2M46 sV2M46 sV2M46];
da179 = [sV2M79 sV2M79 sV2M79];
sgVS = V2S .* dg;

C1M1 = C1M - miu.*(b1*(sV2M10.*da1));
V1M1 = V1M - miu.*(b1*(sV2M10.*da1))*p(:,j)';
C1M2 = C1M - miu.*(b1*g'*da179')';
V1M2 = V1M - miu.*(b1*g'*da179')'*p(:,j)';
C1M3 = C1M - miu.*(b1*sgVS'*da146')';
V1M3 = V1M - miu.*(b1*sgVS'*da146')'*p(:,j)';
C1M4 = C1M - miu.*(b1*ps(:,j)*sgVS'*da113')';
V1M4 = V1M - miu.*(b1*ps(:,j)*sgVS'*da113')'*p(:,j)';
C1M = (C1M1+C1M2+C1M3+C1M4)./4;
V1M = (V1M1+V1M2+V1M3+V1M4)./4;
C2M(1:3) = C2M(1:3) - miu.*(b1*ps(:,j)*sgVS);
C2M(4:6) = C2M(4:6) - miu.*(b1*sgVS);
C2M(7:9) = C2M(7:9) - miu.*(b1*g);
C2M(10) = C2M(10) - miu.*b1;
V2M(1:3,:) = V2M(1:3,:) - miu.*(b1*ps(:,j)*sgVS*a1');
```

```
V2M(4:6,:) = V2M(4:6,:) - miu.*(b1*sgVS*a1');
V2M(7:9,:) = V2M(7:9,:) - miu.*(b1*g*a1');
V2M(10,:) = V2M(10,:) - miu.*(b1*a1)';
end

SSE = sumsqr(Verr);
RMS1= RMS;
RMS = sqrt(SSE/N)
VSSE = [VSSE; RMS];
k = k+1;
if RMS > ssse
   if RMS > RMS1
      loop = k;
      k=nepoch+1;
   end
else
   loop1 = k;
   k=nepoch+1;
   RMStr = RMS
end
end
hstime = toc;

%%%%%%%  testing  %%%%%%%%%%%%%
f=fopen('./fts70','r');
p=[]; ps=[]; t=[];

A=[];
for i = 1:NT
        a=fscanf(f,'%f',[1 9]);
        A=[A;a];          %x1, x2, y will be stored as a row in array A
end
fclose(f);

p = ([A(:,2) A(:,3:4)])';
ps = (A(:,1))';
t = (A(:,6))';

Verr = [];
Z=[];
for j = 1:NT
    a1 = feval(f1,V1M*p(:,j),C1M);  % output of first hidden layer
    a2 = feval(f2,V2M*a1,C2M);      % output of master network
    V1S = a2(1:3); C1S = a2(4:6); V2S = a2(7:9); C2S = a2(10);
    g = feval(f1,V1S*ps(:,j),C1S);  % output of first hidden layer in
slave network
    z = (V2S' * g) + C2S;
    error = (z-t(:,j));
    Z=[Z z];
    Verr = [Verr; error];
end
```

```
SSE = sumsqr(Verr);
MSE = SSE/NT;
RMS = sqrt(MSE)
```

D.6 Sample Program for Output-tuned HSNN

```
clear all
FIRST=18;    % Number of Neurons in the first hidden layer of slave
SECOND=1;    % Number of Neurons in the second hidden layer slave
FM = 6;
SM = 1;
N = 150; % Total number of training data set
miu = 0.02; % Learning rate
nepoch =10000;
NT=70;
RMS = 1;
ssse = 0.04;

% define an input vectors and a target vector y.
f=fopen('./F1th815','r');

A=[];
for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];          %x1, x2, y will be stored as a row in array A
end
fclose(f);

f=fopen('./VyLx.txt','r'); %o/p trained data (V,L,x) fr serial network

for i = 1:N
        a=fscanf(f,'%f',[1 4]);
        AI=[AI;a];          %x1, x2, y will be stored as a row in array
A
end
fclose(f);

F = A(:,3) * 250;
V = AI(:,1) .* F;
L = AI(:,3) .* F;

FP = (V+L)./F;
V = V./FP;
L = L./FP;
z = A(:,4);
x = AI(:,4);
```

```
yp = AI(:,2);

for i = 1:N;
   if yp(i) < 0.02;
      num(i) = 0;
   elseif V(i) < 1;
      num(i) = 0;
   else
      num(i) = ((F(i)*z(i) - L(i)*x(i))/V(i))-yp(i);
   end
end

p = (num);
ps = ([AI(:,1:4) A(:,3:4)])';
t = (A(:,7))';
f1 = 'logsig';
s1 = FIRST;
f2 = 'purelin';
s2 = SECOND;

%Initializing weights and biases
[V1M,C1M] = feval(feval(f1,'init'),FM,p);
[row,col]= size(p);
[V1M,C1M] = rands(FM,row);
x = ones(FM,1) * feval(f1,'output');
[S,Q] = size(SM);
if max(S,Q) > 1, s2 = S; end
[V2M,C2M] = feval(feval(f2,'init'),s2,x);

[V1S,C1S] = feval(feval(f1,'init'),s1,ps);
[row,col]= size(ps);
[V1S,C1S] = rands(s1,row);
x = ones(s1,1) * feval(f1,'output');
[S,Q] = size(s2);
if max(S,Q) > 1, s2 = S; end
[V2S,C2S] = feval(feval(f2,'init'),s2,x);


% PRESENTATION PHASE
VSSE = [];
Vda = [];
tic;
k = 1;
while RMS > ssse
Verr = [];
for j = 1:150
a1 = feval(f1,V1M*p(:,j),C1M);   % output of first hidden layer
a2 = feval(f2,V2M*a1,C2M); % output of master network
g1 = feval(f1,V1S*ps(:,j),C1S);  % output of first hidden layer
g2 = feval(f2,V2S*g1,C2S); % output of slave network

z = (a2(1,:) .* g2); % + a2(2,:);
```

```
error = (z-t(:,j));
Verr = [Verr; error];

% Update V's and C's
da1 = a1 .* (1-a1);      % deriv of a1
b1 = 2 .* error .* ps(:,j);
sV2M = sum(V2M);      % summing up V2M column-wise
dint = sV2M.*da1';

dg1 = g1 .* (1-g1);      % deriv of g1
sV2S = sum(V2S);      % summing up V2M column-wise
dints = sV2S.*dg1';

C1M = C1M - miu.*(2.*error.*g2*dint)';
V1M = V1M - miu.*(2.*error.*g2*dint)'*p(:,j)';
C2M(1,1) = C2M(1,1) - miu.*(2.*error.*g2);
V2M(1,:) = V2M(1,:) - miu.*(2.*error.*g2*a1)';

C1S = C1S - miu.*(2.*error.*a2*dints');
V1S = V1S - miu.*(2.*error.*a2*dints'*ps(:,j)');
C2S = C2S - miu.*(2.*error.*a2);
V2S = V2S - miu.*(2.*error.*a2*g1');

end

SSE = sumsqr(Verr);
RMS1 = RMS;
RMS = sqrt(SSE/N)
VSSE = [VSSE; RMS];
k = k+1;
if RMS > ssse
   if RMS > RMS1
      loop = k;
      k=nepoch+1;
   end
else
   loop1 = k;
   k=nepoch+1;
   RMStr = RMS
end
end
hstime = toc;


%%%%%%%%  testing  %%%%%%%%%%%%%%
f=fopen('./fts70k','r');
p=[]; ps=[]; t=[];

A=[]; AI=[];
for i = 1:N
        a=fscanf(f,'%f',[1 10]);
        A=[A;a];        %x1, x2, y will be stored as a row in array A
```

```
end
fclose(f);

f=fopen('./VyLxts.txt','r');

for i = 1:N
        a=fscanf(f,'%f',[1 4]);
        AI=[AI;a];        %x1, x2, y will be stored as a row in array
A
end
fclose(f);

F = A(:,3) * 250;
V = AI(:,1) .* F;
L = AI(:,3) .* F;
FP = (V+L)./F;
V = V./FP;
L = L./FP;
z = A(:,4);
x = AI(:,4);
yp = AI(:,2);
for i = 1:NT;
   if yp(i) < 0.02;
      num(i) = 0;
   elseif V(i) < 1;
      num(i) = 0;
   else
      num(i) = ((F(i)*z(i) - L(i)*x(i))/V(i))-yp(i);
   end
end
p = (num);
ps = ([AI(:,1:4) A(:,3:4)])';
t = (A(:,7))';

Verr = []; Z=[];
for j = 1:NT
    a1 = feval(f1,V1M*p(:,j),C1M);  % output of first hidden layer
    a2 = feval(f2,V2M*a1,C2M);% output of master network
    g1 = feval(f1,V1S*ps(:,j),C1S); % output of first hidden layer
    g2 = feval(f2,V2S*g1,C2S);    % output of slave network

    z = (a2(1,:) .* g2); % + a2(2,:);
    Z = [Z;z];
    error = (z-t(:,j));
    Verr = [Verr; error];
end

SSE = sumsqr(Verr);
MSE = SSE/NT;
RMS = sqrt(MSE);
```