

FAULT DIAGNOSTIC SYSTEM FOR PREDICTIVE MAINTENANCE ON A BRAYTON CYCLE POWER PLANT

C. Vorster B.ing

**Dissertation submitted in partial fulfillment of the requirements
for the degree Magister in Engineering in Electronic
Engineering of the Potchefstroom University of Christian
Higher Education**

Supervisor: Prof. C.P. Bodenstein

May 2004

Potchefstroom

Acknowledgement

To the One and only Almighty source of motivation, knowledge and love...

Abstract

Model-based fault detection and diagnostic systems have become an important solution (Munoz & Sanz-Bobi, 1998:178) in the industry for preventive maintenance. This not only increases plant safety, but also reduces down time and financial losses. This paper investigates a model-based fault detection and diagnostic system by using neural networks.

To mimic process models, a normal feed-forward neural network with time delays is implemented by using the MATLAB[®] neural network toolbox. By using these neural network models, residuals are generated. These residuals are then classified by using other neural networks. The main process in question is the Brayton cycle thermal process used in the pebble bed modular reactor. Flownet simulation software is used to generate the data, where practical data is absent.

Various training algorithms were implemented and tested during the investigation of modelling and classification concepts on two benchmark processes. The training algorithm that performed best was finally implemented in an integrated concept.

Opsomming

Modelgebaseerde foutopspoor- en diagnoseringstelsels het van kardinale belang geword in die industrie ten opsigte van voorkomende onderhoud (Munoz & Sanz-Bobi, 1998:178). Dit bevorder nie net fabrieksveiligheid nie, maar verminder ook afte en finansiële verliese. Hierdie skripsie ondersoek 'n modelgebaseerde foutopspoor- en diagnoseringstelsel deur van neurale netwerke gebruik te maak.

Om prosesse na te boots, word van normale vorentoevoer neurale netwerke met tydvertragings gebruik gemaak deur middel van MATLAB[®] neurale netwerk gereedskap. Residue word verkry deur van neurale netwerk modelle gebruik te maak. Die residue word dan geklassifiseer deur van ander neurale netwerke gebruik te maak. Die hoof proses onder die vergrootglas is die Brayton-siklus termiese proses wat in die korrelbed modulere reaktor gebruik word. Flownet word gebruik om die proses te simuleer en die nodige data te genereer.

Verskeie opleidingsalgoritmes is toegepas en getoets gedurende die ondersoek van die modellering en klassifiseringkonsepte op twee maatstafprosesse. Die algoritme wat die beste werkverrigting gelever het, word dan uiteindelik geïmplementeer in die geïntegreerde foutopspoor- en diagnoseringstelsel.

Table of Contents

Acknowledgement	I
Abstract.....	II
Opsomming	III
Table of Contents.....	IV
List of Figures.....	VI
List of Tables	VIII
List of Abbreviations	IX
List of symbols	X
1 INTRODUCTION	1
1.1 Background.....	2
1.2 Problem statement	2
1.3 Proposed solution	3
1.4 Specific problems	3
1.5 Methodology.....	4
1.6 Research overview.....	4
2 FAULT DIAGNOSTICS AND RELATED RESEARCH.....	6
2.2 Diagnostic Methods	7
2.3 Some existing research	8
2.3.1 The radial basis function neural networks (RBF).....	9
2.3.2 The probabilistic radial basis function network (PRBFN)	11
2.3.3 Kalman filters	14
3 NEURAL NETWORK THEORY.....	16
3.1 Background.....	17
3.2 Network topologies.....	19
3.2.1 Multilayer feed-forward networks.....	19
3.2.2 Recurrent networks.....	23
4 RESIDUAL CLASSIFICATION	25
4.1 Problem discussion	26
4.2 Detectability and isolability	27
4.3 Methodology.....	28
4.3.1 SIMULINK®	28
4.3.2 The data	29
4.3.3 Neural network architectures.....	31
4.3.3.1 Three-network topology	32
4.3.3.2 Single-network topology	32
4.4 Results	32
4.4.1 Three-network topology	32
4.4.2 Single-network topology	35
4.4.3 Multi-input single-output system.....	37
4.4.4 Noisy data	38
4.4.5 Quantised data	39
4.4 Conclusion	40

5	INDUCTION MACHINE BENCHMARK.....	42
5.1	Purpose	43
5.2	Background.....	43
5.3	Methodology.....	46
5.3.1	Simulation.....	46
5.3.2	Training data.....	47
5.3.3	Neural network topology	49
5.4	Results	50
5.5	Conclusion	53
6	FLOWNET MODEL OF THE BRAYTON CYCLE.....	54
6.1	Background.....	55
6.1.1	Introduction	55
6.1.2	Theory.....	55
6.1.2.1	The core	56
6.1.2.2	The compressors	57
6.1.2.3	The turbines	59
6.1.2.3	The heat exchanger.....	60
6.1.2.5	Flownet	61
6.2	Methodology.....	62
6.2.1	Modelling.....	62
6.2.1.1	Brayton cycle.....	62
6.2.1.2	FDD system	63
6.2.1.2	Modelling results	65
6.2.2	Classification	67
6.2.2.1	Concept.....	67
6.2.2.2	Classification of data	67
6.2.2.3	Classification results.....	68
6.2.3	Decision logic	72
6.2.3.1	Methodology.....	72
6.2.3.2	Applied logic	73
6.2.3.3	Integrated concept.....	74
6.2.3.4	Decision logic results.....	75
6.3	Measured data.....	76
6.4	Conclusion	77
7	CONCLUSION	78
7.1	Conclusion	79
8	APPENDIX	81
	REFERENCES	87

List of Figures

Figure 1.1: The Brayton cycle as used in the pebble bed modular reactor.....	3
Figure 2.1: Diagnostic family tree.....	8
Figure 2.2: Schematic representation of an FDD system.	9
Figure 2.3: A generic fault detection method.	10
Figure 2.4: PRBFN architecture.	13
Figure 2.5: Fault detection by using Kalman filters.	15
Figure 3.1: An actual neuron.	17
Figure 3.2: A mathematical equivalent of the neuron.	18
Figure 3.3: A multilayer feed-forward network architecture.	20
Figure 3.4: The tansig (left) and purelin (right) activation functions.	21
Figure 3.5: The log-sigmoid activation function.	21
Figure 3.6: Information window while training a MATLAB® neural network.	22
Figure 3.7: Basic recurrent network architecture.	24
Figure 4.1: The parallel processes in SIMULINK®	26
Figure 4.2: Input to the classification networks.	27
Figure 4.3: The input to the SIMULINK model.....	29
Figure 4.4: Signals representing a bias (left) and gain (right) fault.....	30
Figure 4.5: Fault windows for bias (left) and gain (right).	30
Figure 4.6: Time performance of the different topologies.	34
Figure 4.7: Accuracy of the different topologies.....	34
Figure 4.8: A radar plot of the cumulative errors over 8 runs.	34
Figure 4.9: Time average plots of different topology groups.....	36
Figure 4.10: Cumulative error of the different topology groups.	36
Figure 4.11: Three similar possibilities that increase classification complexity.	37
Figure 4.12: Neural network uncertainty with similar residuals.	38
Figure 4.13: The same network with no noise and 13 dB SNR noise.	39
Figure 4.14: 2-bit quantized data.	40
Figure 5.1: The rotor cage of a squirrel-cage induction machine.....	43
Figure 5.2: A SIMULINK® model of the induction machine.	47
Figure 5.3: An input/output pair sample of the training data.	48
Figure 5.4: A feed-forward neural network with time delays [input at $t_0, t_1, \dots$].	49
Figure 5.5: Same inputs (red) with different outputs (blue).	51
Figure 5.6: Neural network simulation with insufficient delays.	51
Figure 5.7: The step response of the 30-20-1 network.	52
Figure 5.8: The step response of the 12-10-1 network.....	53
Figure 6.1: Schematic representation of a compressor.....	58
Figure 6.2: T-s diagram for a compressor.	58
Figure 6.3: Schematic representation of a turbine.	59
Figure 6.4: T-s diagram for a turbine.	60
Figure 6.5: Schematic representation of a heat exchanger.	60
Figure 6.6: Flownet model schematic.....	61
Figure 6.7: Input/output samples.	62

Figure 6.8: FDD concept on part of the Brayton cycle.	63
Figure 6.9: Three neural network layout combinations.....	64
Figure 6.10: The modelling capabilities of these neural networks.....	65
Figure 6.11: Illustration of insufficient weights error for this scenario.....	66
Figure 6.12: An example of sensor fault residuals.	68
Figure 6.13: Misclassifications of the first classification network, C1.	69
Figure 6.14: Output of a 2-layer topology using both training sets.....	70
Figure 6.15: Classification network trained with both training sets.....	71
Figure 6.16: MATLAB® GUI developed to integrate all the FDD concepts.....	74
Figure 6.17: An example of a sensor 1 and 3 fault with sensor 3 detected first.....	75
Figure 6.18: Load rejection test of the PBMM.....	76
 Figure A.1: Schematic illustration of the multi-input single output model.....	 86

List of Tables

Table 3.1: Commonly used MATLAB [®] training functions.	20
Table 4.1: Possible outputs of the residual neural networks.	31
Table 4.2: The output target format for the single-network scenario.	31
Table 4.3: Performance comparison of the three-network topology.	33
Table 4.4: Performance comparisons of the single-network topology.	36
Table 4.5: Cumulative MSE for quantised test set.	39
Table 5.1: Symbol definitions.	45
Table 5.2: Induction machine parameter definitions.	45
Table 5.3: Complex networks are insufficient.	52
Table 6.1: Symbol definitions for the Brayton-cycle model.	56
Table 6.2: Accuracy (MSE) and speed of the three training algorithms investigated.	65
Table 6.3: Effect of the extra training set on classification performance.	70
Table 6.4: Single sensor error truth table.	73
Table 6.5: Multiple sensor error logic table.	73
Table A.1: Performance data for the 3-network topology.	82
Table A.2: Performance data for the single-network 2-layer architecture.	83
Table A.3: Performance data for the single-network 3-layer architecture.	85
Table A.4: Performance with noise added for the three-network 6-3 architecture.	85

List of Abbreviations

FDD	Fault detection and diagnosis
GUI	Graphic user interface
HTR	High temperature reactor
MISO	Multi-input single-output
MNN	Multilayer neural network
MSE	Mean-square-error
OLS	Orthogonal least squares
PBMM	Pebble bed micro model
PBMR	Pebble bed modular reactor
PDF	Probability density function
PRBFN	Probabilistic radial basis function network
RBF	Radial basis function
SISO	Single-input single-output
SNR	Signal to noise ratio

List of symbols

Name	Description	Units
Chapter 3		
u, x	System input related variables	-
y	System output related variable	-
w	Neural network connection weights or model parameters	-
ξ	Centre vector	-
d	Actual output of a parameter	-
X_i, x	State space variables	-
e	Error or residual	-
a	Activation functions	-
A, B, C	Constant matrices	-
K_i	i -th kalman filter gain	-
Chapter 5		
v_a, v_b, v_c	Stator phase-voltages	V
i_a, i_b, i_c	Stator phase-currents	A
$\bar{v}_s(t)$	Stator voltage space-vector	V
$\bar{i}_s(t), \bar{i}_r(t)$	Stator and rotor current space-vectors	A
v_{sd}, v_{sq}	Stator voltage vector components in rotational reference frame	V
$v_{s\alpha}, v_{s\beta}$	Stator voltage vector components in stator oriented frame	V
i_{sd}, i_{sq}	Stator current vector components in rotational reference frame	A
$i_{s\alpha}, i_{s\beta}$	Stator current vector components in stator oriented frame	A
i_{rd}, i_{rq}	Rotor current vector components in rotational reference frame	A
$\bar{\lambda}_s(t), \bar{\lambda}_r(t)$	Stator and rotor flux space-vectors	
$\lambda_{sd}, \lambda_{sq}$	Stator flux vector components in rotational reference frame	
$\lambda_{rd}, \lambda_{rq}$	Rotor flux vector components in rotational reference frame	
ω_d	Synchronous angular speed	
ω_{dA}	Slip angular speed	
ω_m	Rotor (mechanical) angular speed	
$\theta_d, \theta_{dA}, \theta_m$	Angles in radian, corresponding to the above speed types	

Name	Description	Units
	Chapter 5	
P_n	Motor rated power	W
I_n	Rated phase current (rms value)	A
V_n	Rated phase voltage (rms value)	V
P	Number of pole-pairs	-
f_n	Rated frequency	Hz
T_n	Rated torque	Nm
T_L	Load torque	Nm
J_{eq}	Moment of inertia (equivalent)	Kg m ²
R_s	Stator resistance	Ω
R_r	Rotor resistance	Ω
L_m	Magnetizing inductance	H
L_{ls}	Stator leakage inductance	H
L_{lr}	Rotor leakage inductance	H
$L_s=L_{ls}+L_m$	Stator equivalent inductance	H
$L_r=L_{lr}+L_m$	Rotor equivalent inductance	H
	Chapter 6	
Q_n	Reaction power	Watt
m, m_a, m_b	Mass flow rate	kg/s
c_p, c_a, c_b	Specific heat capacitance of fluid at constant pressure	J/kgK
T_1, T_{a1}, T_{b1}	Inflow temperature	K
T_2, T_{a2}, T_{b2}	Outflow temperature	K
T_a	Ambient temperature	K
R_a	Thermal resistance to ambient	W/K
P_1	Inflow pressure	Pa
P_2	Outflow pressure	Pa
C, C_a, C_b	Thermal capacitance	K/J

FAULT DIAGNOSTIC SYSTEM FOR PREDICTIVE MAINTENANCE ON A BRAYTON CYCLE POWER PLANT

1 INTRODUCTION

A compact background discussion of the motivation for this research is given. The problem that is addressed in this paper is stated, followed by a proposed solution. Specific problems are highlighted. A methodological approach is proposed followed by an overview of the issues addressed in the different chapters.

FAULT DIAGNOSTIC SYSTEM FOR PREDICTIVE MAINTENANCE ON A BRAYTON CYCLE POWER PLANT

1.1 Background

One of the two major factors forcing us to look at preventive maintenance is the safety of humans. Should an explosion occur due to a fault in a process, severe injuries or even fatalities could be sustained. To prevent such a scenario is of the utmost importance. The second major factor that contributes to the necessity for predictive maintenance is the financial factor. If the incipient failure of a component were detected in time, this component could be replaced during a planned shut-down. Catastrophic failure could cause failure of other components, depending on the severity of the consequential damage. By using fault detection and diagnostics (FDD), unscheduled down time could be reduced as well as damage to other components during failure. The financial implications are also of paramount importance, as Barringer and Woodrow (2002) point out that failures represent loss of money.

1.2 Problem statement

The above-mentioned factors stress the necessity of being able to predict failure of components or a faulty process status. The aim of this thesis is to investigate a fault detection and diagnostic (FDD) concept that could detect and recognise faulty behaviour of a process. A few known processes will be investigated before progressing to the Brayton thermal energy conversion cycle. The properties of neural networks will be put to the test during the investigation of an FDD system. For the diagnostic part, the classification capabilities are tested while the modelling capabilities of neural networks are tested in the detection part.

Detection of faults must ideally occur early enough so that actual failure of the process could be prevented. The Brayton cycle is a thermal energy-conversion process that is used in industrial applications to convert thermal energy to mechanical energy. The Brayton cycle is also implemented in the Pebble bed modular reactor (PBMR) nuclear power plant concept. See figure 1.1 (http://www.pbmr.co.za/2_about_the_pbmr/2_3how_it_works.htm). Since the PBMR has not yet been built, data is obtained from a simulation of the plant in a software package called Flownet where a model of the proposed plant was developed.

FAULT DIAGNOSTIC SYSTEM FOR PREDICTIVE MAINTENANCE ON A BRAYTON CYCLE POWER PLANT

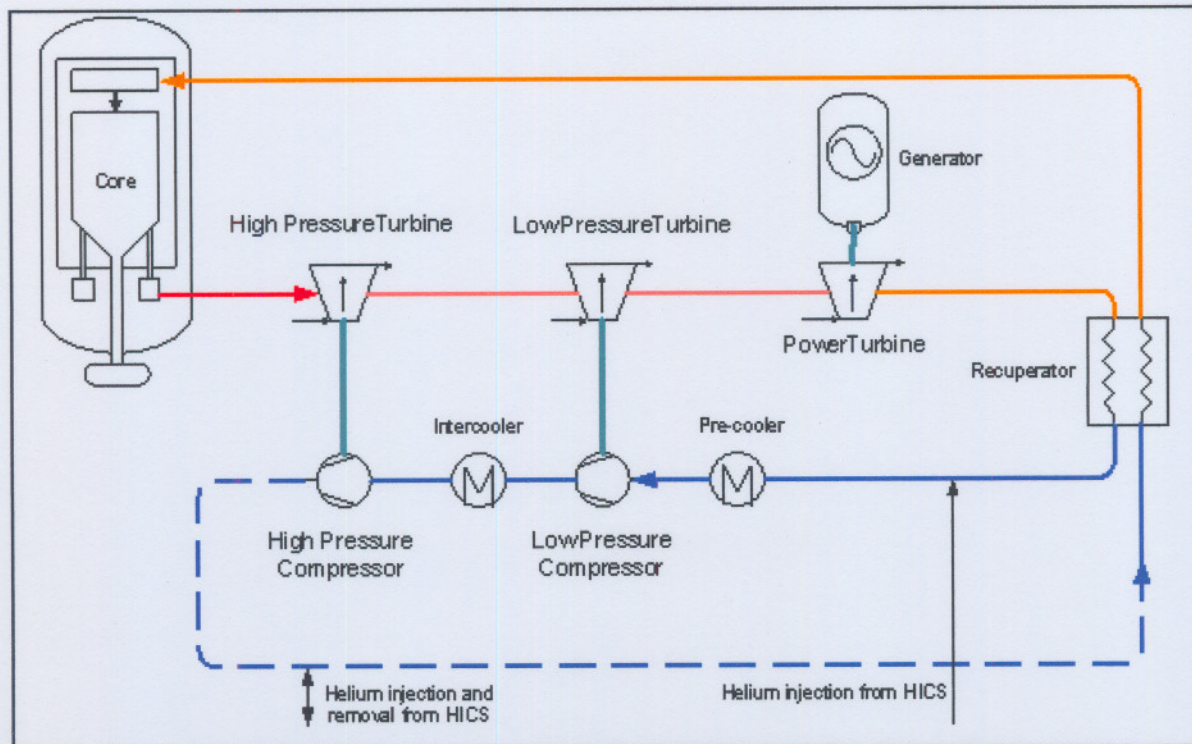


Figure 1.1: The Brayton cycle as used in the pebble bed modular reactor.

1.3 Proposed solution

A neural network, of which a discussion is given later, is implemented to firstly model the process response and secondly, by using another neural network, to classify identified errors in the process. The principles of fault detection and diagnostics will be discussed in Chapter 2. MATLAB[®] has a neural network toolbox that simplifies the process of investigating the different neural network topologies. The feasibility of the FDD concepts are investigated by implementing it on benchmark models to gain an understanding of the underlying principles. The general conclusions obtained from the simulations are then applied to a model of the PBMR.

1.4 Specific problems

- Diagnosing faults by using neural networks to classify residuals;
- modelling a process using neural networks for fault detection;
- Integrating the above mentioned solutions;
- Successfully implementing the FDD concept on the simulated pebble bed micromodel, PBMM.

FAULT DIAGNOSTIC SYSTEM FOR PREDICTIVE MAINTENANCE ON A BRAYTON CYCLE POWER PLANT

1.5 Methodology

A major part in the development of the FDD will be the investigation of neural networks. These networks must be able to identify and classify faulty behaviour in the response data. Although multiple errors are discussed the focus is mainly on single errors. A discussion of neural networks is given in Chapter 3. MATLAB® will be used to develop this system because it has a neural network toolbox that makes development relatively easy. Finding the neural network topology and learning algorithm that generate the best results will be a major part of the work. Published research results will be consulted to find the appropriate neural network topology.

Data needed to train the networks will be obtained by means of simulation. This data could be manipulated to gain the most efficient results. To establish the tendencies of the data one is working within, might prove to be equally important.

When the concepts have been developed on the benchmark processes and the experimental results prove to be satisfactory, the Flownet model's simulated outputs will be used to train the network. The benchmark processes in this case are transfer function models and a model of an induction machine.

1.6 Research overview

Chapter 2: Background theory of fault detection and diagnostic systems and a literature survey of what has already been done.

Chapter 3: Theory of neural networks as applicable to this investigation.

Chapter 4: Investigation of neural networks as fault identifiers. Residuals are obtained by implementing two transfer function models that run parallel, one without errors and the other with errors. These transfer function models are implemented in SIMULINK®.

FAULT DIAGNOSTIC SYSTEM FOR PREDICTIVE MAINTENANCE ON A BRAYTON CYCLE POWER PLANT

Chapter 5: Modelling a dynamic process with neural networks. An induction machine was chosen to serve as a benchmark for a dynamic non-linear system. The model was developed in SIMULINK and the neural network toolbox of MATLAB® is used to implement the neural network. This is a known process with some well-known faults and symptoms.

Chapter 6: Implementing the integrated concept on a FLOWNET model of the PBMM.

Chapter 7: A final conclusion on this research is given.

2 FAULT DIAGNOSTICS AND RELATED RESEARCH

A background is given on fault diagnosis and diagnostic methods. The difficulties that are faced when developing a fault detection and diagnostic system are discussed. An overview is given of some related research that has been done.

FAULT DIAGNOSTICS AND RELATED RESEARCH

2.1 Fault diagnosis

The need to detect and diagnose faulty behaviour in a process, especially in its early stages, has been mentioned. A brief discussion of some practical challenges and some attributes of a diagnostic system are now given.

Process dynamics, lack of adequate models, incomplete and uncertain data and diverse sources of knowledge are amongst others, factors that make predictive fault detection and diagnosis (FDD) challenging (Dash & Venkatasubramanian, 2000). In industry the wide variety of processes leads to an equally wide variety of control systems that exist. This also applies to fault diagnostic systems.

For a diagnostic system to be an effective tool, such a system should have certain attributes. Foremost is the fact that the system should be able to detect and diagnose the fault as *early* as possible to be effective. When the system detects a faulty situation, it should be able to *discriminate* between various possible failures. However, the degree of *isolability* must not be so high that the system sees modelling uncertainties as faulty behaviour. The *robustness* of the system to noise and uncertainties will allow the system performance to *degrade gradually*, instead of falling away suddenly.

An important attribute is an explanation facility that would give possible origins and causes of the identified process faults. This facility would need a very descriptive set of historical data or experienced know-how. When the process is updated or changed the system should be able to adapt to. These changes should include minor changes, such as changes in the environment.

2.2 Diagnostic Methods

There are various diagnostic methods from which to choose. However, it is not that simple: an application might only allow certain types of methods. In figure 2.1 (Dash & Venkatasubramanian, 2000) the different methods are shown in a schematic diagram to illustrate the separate directions one can follow. Combining different concepts is sometimes necessary to get a robust and generalising system. The two main differences between these methods relate to methods that use process model information and those that use process history data.

FAULT DIAGNOSTICS AND RELATED RESEARCH

With the process model-based method a fundamental understanding of the process is needed. This is used to describe the interactions of various process variables and input and output relations. The process history-based method uses, as the name implies, historical data that has been collected from an existing process. Each of these methods is then again divided into quantitative and qualitative methods as shown in figure 2.1.

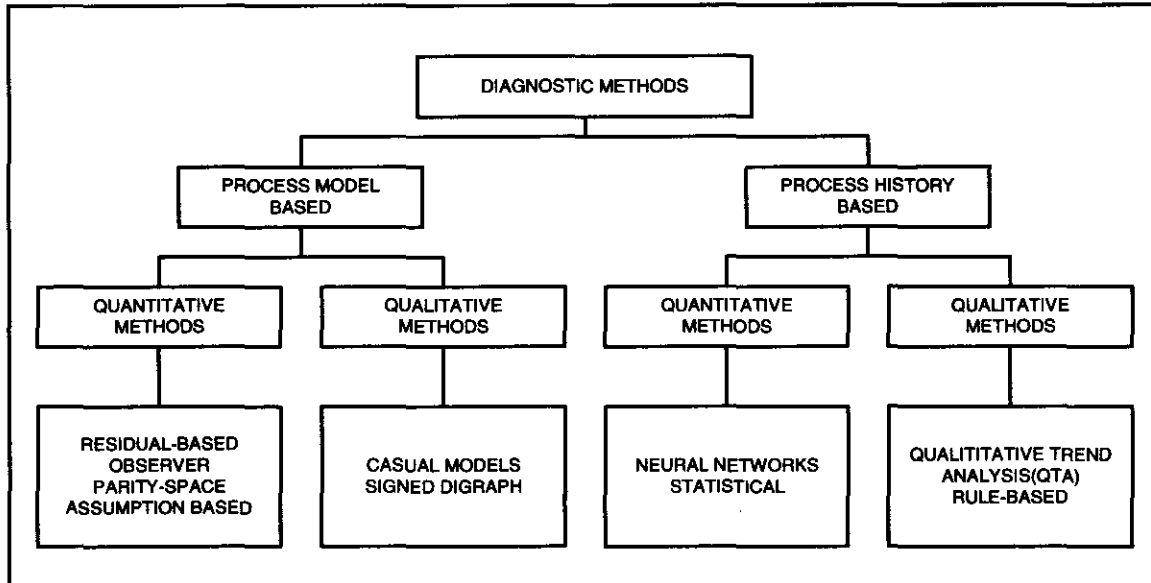


Figure 2.1: Diagnostic family tree.

Neural networks are shown as process history-based quantitative methods. In this investigation the process history will be obtained by simulation, since the physical plant has not yet been built. This investigation also shows whether MATLAB[®] and its neural network toolbox will suffice to provide the core of a diagnostic system.

2.3 Some existing research

Although there are FDD systems that use statistics and methods other than neural networks, the emphasis in this work is on neural networks. The reason for the popularity of neural networks is the generalising ability for non-distinct behaviour of a system (Fuente & Vega, 1999). A few applications of interest are discussed below.

2.3.1 The radial basis function neural networks (RBF)

Chen and Lee (2001) use radial basis function (RBF) networks for online state estimation, trained with the orthogonal least squares (OLS) algorithm. To train for detection and diagnosis, they use the back-propagation algorithm. They investigated fault detection and diagnosis on an airplane so that the flight control system could monitor the plane. When the system detects a fault on a component, the flight control system reacts appropriately to prevent an accident.

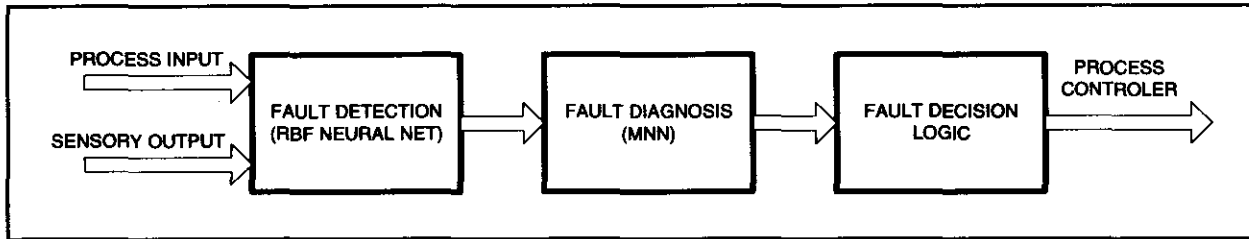


Figure 2.2: Schematic representation of an FDD system.

Figure 2.2 illustrates how they implemented a fault detection and diagnostic system. The two major parts of their system are *detecting* a fault by using the RBF network and the *classification* of the fault by using the multilayer neural network (MNN). The RBF uses less computation time (Moody & Darken, 1989) during training than topologies like the MNN and is also a more compact architecture (Lee & Kil, 1991).

Apart from that, RBF networks are highly suited for non-linear approximation. Using data that is taken from the system itself or a simulation of that system, the RBF network is trained to act like the system. A model of the process that is as realistic as possible is used to generate data for fault scenarios that are not available in the real process.

The RBF network approach in representing time-varying non-linear dynamic system is to use a combination of system input and output with some time-delay units. It is assumed that a time-varying non-linear dynamic system is described by:

$$y(k) = f(y(k-1), \dots, y(k-n_y), u(k-d-1), \dots, u(k-d-n_u)) \quad (2.1)$$

with f an unknown non-linear function, u and y are the system's input and output and n_y , n_u and d represents time delays and orders of the model respectively. By using an RBF network that has one hidden layer and one output layer, its response can be expressed as:

FAULT DIAGNOSTICS AND RELATED RESEARCH

$$\hat{y} = f(x) = \sum_{\alpha} w_{\alpha} \phi(\|x - \xi_{\alpha}\|) \quad (2.2)$$

Here ϕ can be selected from many radial functions and in general this radial function is chosen as the Gaussian function. ξ_{α} is the center vector of the α -th hidden unit and w_{α} is the connection weight between the α -th hidden unit and one output layer. This RBF network becomes a function approximation problem of the type

$$\hat{y}[k] = f(x[k], n) \quad (2.3)$$

that will simulate the system.

The detection component of the FDD system generates residuals by comparing the actual system variables with those simulated by the RBF network. If no faulty behaviour occurs, the residuals that are generated will originate from noise and system disturbances. There is a threshold that must be exceeded before a residual qualifies to indicate a fault in the system. This threshold should be chosen so that it will not wrongly mistake noise for a failure, or such that only extreme faults are detected with many misses on less severe faults.

Residuals generated by the system are stored in a database and serve as training data. The multilayer perceptron neural network implements the back-propagation learning algorithm for learning to classify these residuals. The aim is to classify the faults with a certain degree of accuracy.

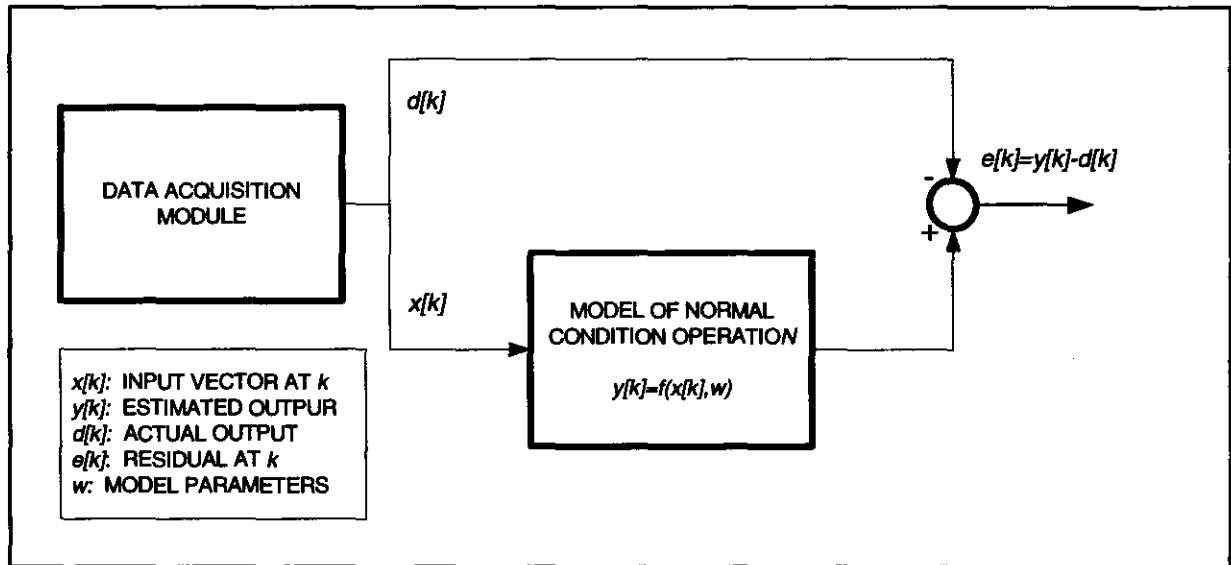


Figure 2.3: A generic fault detection method.

2.3.2 The probabilistic radial basis function network (PRBFN)

The detection method investigated by Munoz and Sanz-Bobi (1998) is discussed. This method is illustrated in figure 2.3. This fault detection method is based on a connectionist characterisation of the normal behaviour of a system or subsystem. A probabilistic radial basis function network (PRBFN) is used as a function approximator in the form of a dynamic black box model. Analytical redundancy is applied for this characterisation. The definition of the reliable domain of the model generally improves model-based fault detection methods. This means that by restricting the residuals generated within practical boundaries, uncertainties decrease.

Limit checking of individual plant variables is another method used for fault detection. However, the thresholds must be set conservatively so that all the normal conditions of the plant are covered. Fault isolation becomes intricate because a single fault may cause many plant variables to exceed their limits. Here pattern recognition was chosen for their flexibility (Munoz & Sanz-Bobi, 1998).

There are pattern-recognition techniques such as Bayes classifiers (Rengaswamy & Venkatasubramanian 2000) or Bayes' classification by using the K-nearest neighbour algorithm (<http://www.eece.unm.edu/controls/papers/ACC97-paper1.pdf>) that is not neural network based. However, as mentioned before, the generalising ability of neural networks has made them popular. The generation of residuals to classify process faults, predominate recent literature.

When using state equation or transfer functions for simulation models, there are three different strategies for residual generation: parity equations, the diagnostic observer and the Kalman filter. Keep in mind that transfer functions only apply to linear systems. Firstly the variables $\{X_1, X_2, \dots, X_{n+1}\}$ are said to be representative of the state of the component, if under normal operation it is possible to define a set of parity relations of the form

$$G_i(X_1, X_2, \dots, X_{n+1}) = 0, \quad i = 1, \dots, m, \quad (2.4)$$

which shows the relations between the lags of the state variables X_j . When a fault is detected at least one of the equations will not be satisfied. By selecting one variable of each G_i as the output variable of the model, the dynamic system can be modelled. Assume that the parity relations G_i ($i=1, \dots, m$) can be expressed as

$$d_i[k] = g_i(d_i^{(k-1)}, u_i^{(k)}, \varepsilon_i^{(k-1)}) + \varepsilon_i[k] \quad (2.5)$$

FAULT DIAGNOSTICS AND RELATED RESEARCH

where $d_i[k] \in \mathfrak{R}$ is the value of the selected output variable for the i th parity equation at time k (consider as present time), $d_i^{(k-1)}$ the vector containing lagged or past values of $d_i[k]$, $u_i^{(k)}$ is a vector containing present and past values of the remaining n input state variables, and $\varepsilon_i[k]$ is a white-noise process.

If $d_i[k]$ can be estimated by an unbiased non-linear model

$$y_i[k] = f_i(d_i^{(k-1)}, u_i^{(k)}, e_i^{(k-1)}) \quad (2.6)$$

then the estimation errors

$$e_i[k] = d_i[k] - y_i[k], \quad i = 1, \dots, m \quad (2.7)$$

are the residuals that one uses to diagnose the fault (Munoz & Sanz-Bobi, 1998).

Another important issue is what a reliable domain of the model would be. The black box model is realised by fitting a function approximator to a set of the input/output relations to act as training data. The reliable area would then be inside the input space of $X \subset \mathfrak{R}^n$. This is called the reliable domain of the model. The PRBFN gives an estimation of the probability density function (pdf), $p_x[k]$ of the input vector $x[k]$. This pdf of $x[k]$ is a good representation of the environment of the training set and thus a good characterisation of the residual $e[k]$. If $p_x[k]$ is low, the reliability of the estimated $e[k]$ is also low.

When to take notice of a residual is another question that needs to be answered. A proposal is to estimate the standard deviation, $s_e[k]$ of the residual as a function of the input vector (Munoz & Sanz-Bobi, 1998). A residual is significantly high if its absolute value exceeds the residual upper bound

$$e_{\max}[k] = 2 \cdot s_e[k] \quad (2.8)$$

The structure of the PRBFN is illustrated in figure 2.4 (Munoz & Sanz-Bobi, 1998). The output could take on one of two forms. For a given input vector $x \in \mathfrak{R}^n$ with its desired output $d \in \mathfrak{R}$, a joint pdf type of expression is obtained given by equation 2.9 (Munoz & Sanz-Bobi, 1998).

$$p(x, d) = \frac{1}{h} \sum_{k=0}^h \left[\left(\prod_{k=0}^n \frac{1}{\sqrt{2\pi} |\sigma_{ik}|} \right) \exp \left(-\sum_{k=1}^n \frac{1}{2\sigma_{ik}^2} (x_k - r_{ik})^2 \right) \right] \times \exp \left(-\frac{1}{2\sigma_{io}^2} (d - v_i)^2 \right) \quad (2.9)$$

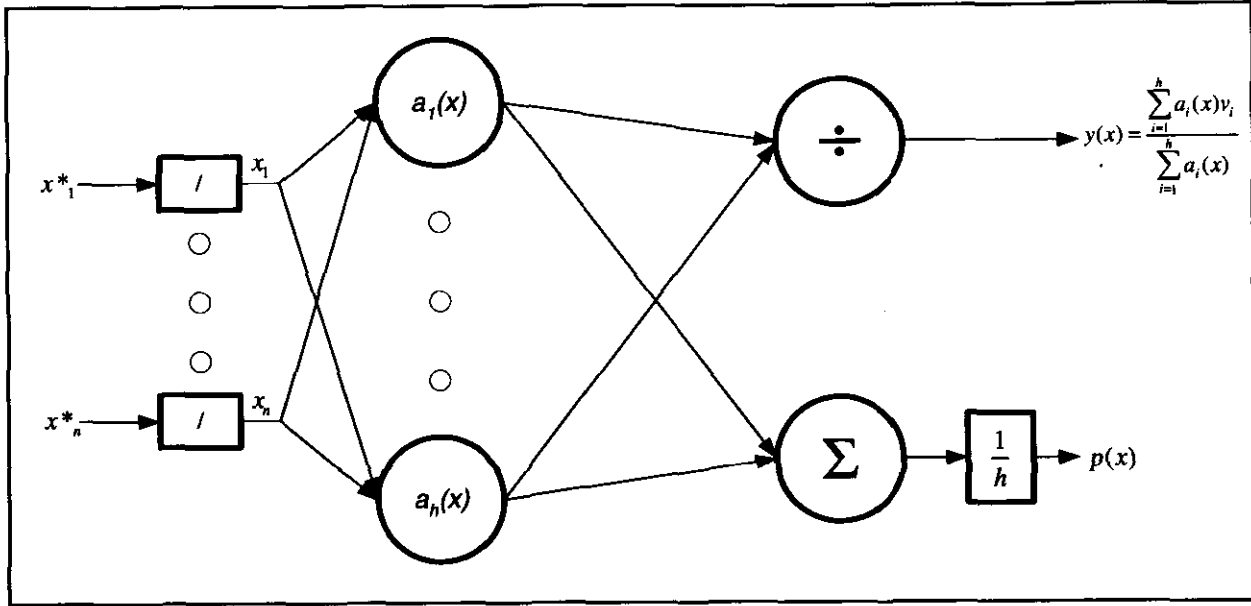


Figure 2.4: PRBFN architecture.

Assume that $p(\mathbf{x}, d)$ is an estimation of the underlying joint pdf. This estimator can be used as a function approximator of the input/output mapping $\mathbf{x} \mapsto d$ of the system. By applying the general regression principle it is seen in figure 2.4 that

$$y(\mathbf{x}) = E\left(\frac{d}{\mathbf{x}}\right) = \frac{\int_{-\infty}^{+\infty} d \cdot p(\mathbf{x}, d) dd}{\int_{-\infty}^{+\infty} p(\mathbf{x}, d) dd} = \frac{\sum_{i=1}^h a_i(\mathbf{x}) v_i}{\sum_{i=1}^h a_i(\mathbf{x})}. \quad (2.10)$$

The activations a_i are given by

$$a(\mathbf{x}) = \left(\prod_{k=1}^n \frac{1}{\sqrt{2\pi} |\sigma_{ik}|} \right) \exp\left(-\sum_{k=1}^n \frac{1}{2\sigma_{ik}^2} (x - r_{ik})^2\right) \quad (2.11)$$

and all this can be structured as a two-layer neural network that gives rise to the PRBFN in figure 2.4. This network can be trained either as a function approximator or an estimator of the pdf of the input vector or it can be trained for both these functions, depending on the learning algorithm. A low-memory quasi-Newton method and cross validation can be used to optimise this procedure (Munoz & Sanz-Bobi, 1998).

2.3.3 Kalman filters

Simani and Fantuzzi (2000) use an FDD system similar to that used in section 2.3.2. Neural networks are used to classify the residuals that are generated from the sensors. The process is assumed to be described by a discrete time, time-invariant linear dynamic model of the type

$$\begin{aligned} x(t+1) &= Ax(t) + B\hat{u}(t), \\ \hat{y}(t) &= Cx(t), \quad t = 1, 2, \dots, \end{aligned} \quad (2.12)$$

where $x(t) \in \mathbb{R}^n$ is the state vector and the output vector is given by $\hat{y}(t) \in \mathbb{R}^m$ with the control input vector $\hat{u}(t) \in \mathbb{R}^r$. The constant matrices A, B and C are obtained by modelling techniques or detection procedures. The actual measurements would then be given by

$$\begin{aligned} u(t) &= \hat{u}(t) + \tilde{u}(t) + f_u(t), \\ y(t) &= \hat{y}(t) + \tilde{y}(t) + f_y(t), \end{aligned} \quad (2.13)$$

Here $\tilde{u}(t)$ and $\tilde{y}(t)$ represent the noise in the sensors that are normally modelled as white-noise Gaussian processes. The last term in each of these equations, $f_u(t)$ and $f_y(t)$, are signals that assume non-zero values when faults are present in the process.

Kalman filters are used to estimate signals that are compared to the actual signals from the sensors. Figure 2.5 (Simani & Fantuzzi, 2000) illustrates the method of using classical Kalman filters to generate residuals. Take notice that each of the m output sensors has its own filter, which simplifies the detection of faults. This situation is, however, not the same for the input sensor values, because every filter receives them all.

The time-invariant, discrete time, linear dynamic system referred to earlier has for the i th Kalman filter the structure (Jazwinski, 1970)

$$\begin{aligned} x_F^i(t+1|t) &= A(I - K_i(t)C_i)x_F^i(t|t-1) + Bu(t) + AK_i(t)y(t), \\ y_F^i(t|t) &= C_i(I - K_i(t)C_i)x_F^i(t|t-1) + C_iK_i(t)y(t). \end{aligned} \quad (2.14)$$

The one-step prediction of the i th component of the state $x(t)$ is given by $x_F^i(t+1|t)$. The estimate of the i th component of the output, $y_i(t)$ is given by $y_F^i(t|t)$, while C_i is the i th row of the output distribution matrix C . Simani and Fantuzzi (2000) use a Riccati equation to compute the time-variant gain of the filter, $K_i(t)$, using the covariance matrix of the output vector noise $\tilde{u}(t)$ and the variance of the i th component of the output noise $\tilde{y}(t)$.

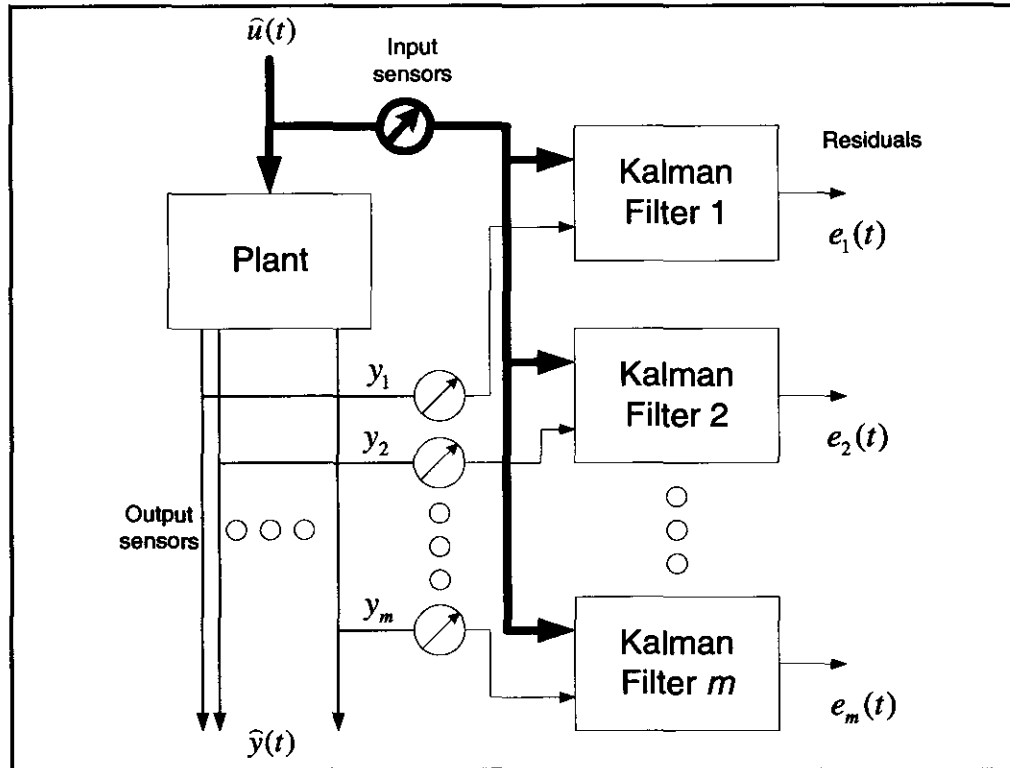


Figure 2.5: Fault detection by using Kalman filters.

The problem, however, is the fact that the process needs to be described with a discrete time, time-invariant dynamic model as described above. This is unnecessary for neural networks, which is another advantage. There are many way to implement the FDD concept, but the basic idea remains the same.

3 NEURAL NETWORK THEORY

The basic theory of neural networks is discussed with the emphasis on how to use the MATLAB[®] neural network toolbox.

NEURAL NETWORK THEORY

3.1 Background

The basic building block of the brain is the neuron, figure 3.1 (<http://faculty.washington.edu/chudler/cells.html>). These neurons are connected in networks with their synaptic terminals connected to the dendrites of other neurons. A network can also have feedback; one example is when the synaptic terminal of a neuron is connected to its own dendrites. There are countless types of network architectures made of basic feed-forward and feed-back network topologies. These two basic networks are discussed later on.

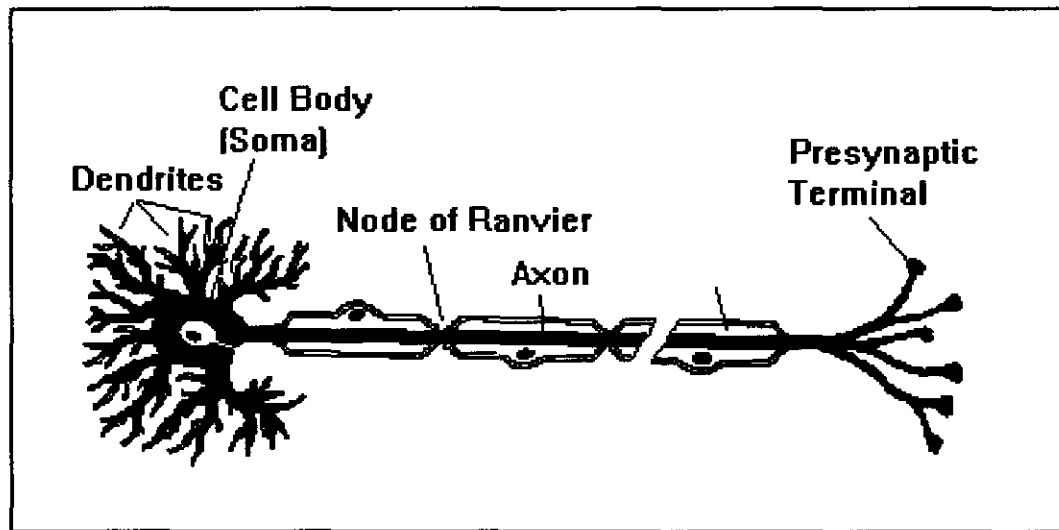


Figure 3.1: An actual neuron.

The connection strengths of the synapses lie in the chemistry of the synaptic terminal and these strengths are manipulated when the brain is learning something. The brain learns from inputs and responses. An example is when a child looks at an *a*, writes it down and tries to remember that it sounds like *a* (/e/), the desired output, to be able to use it later on. The knowledge is imbedded within the network structure via the synaptic connection strengths. The human brain is a highly parallel-operating system. This characteristic makes it ideal for recognising patterns like faces or any image for that matter almost immediately (Haykin, 1999).

Neural networks attempt to simulate some functions of the brain, which would have useful computational advantages. A more descriptive definition is given by Haykin (1999):

A neural network is a massively parallel-distributed processor consisting of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

NEURAL NETWORK THEORY

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

The learning process is an algorithm and quite a number of these were developed, of which some will be discussed later on. The function of these algorithms is to change the synaptic weights between the neurons to integrate within the network, the knowledge it is being *fed*.

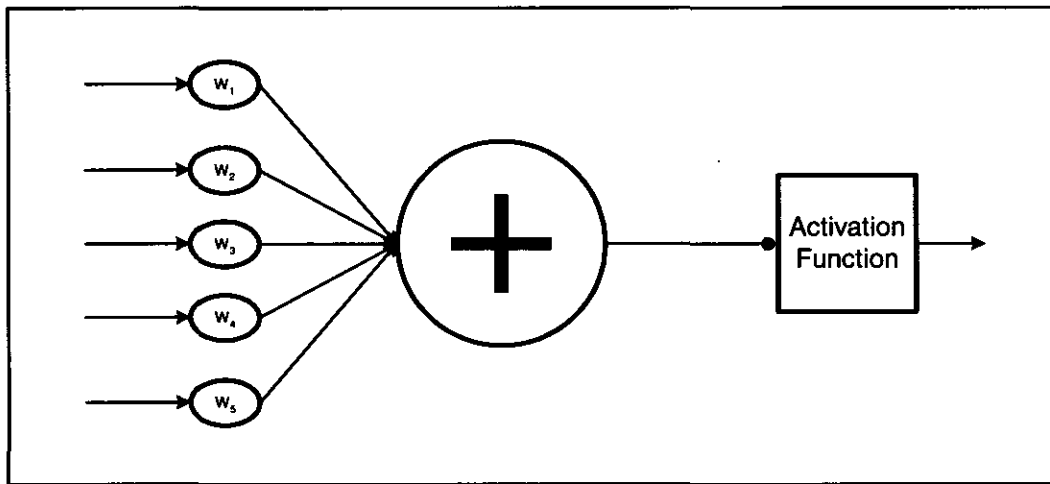


Figure 3.2: A mathematical equivalent of the neuron.

Figure 3.2 provides a mathematical representation of a neuron with the weights representing the synaptic connection strengths. In equation form:

$$Output = f\left(\sum_{n=1}^5 i_n W_n\right), \quad (3.1)$$

where f represents the activation function and i_n the input values. This is only a simplified representation of a real neuron found in the human brain. A bias, which acts like an input with a fixed value, may also be added. The bias has a weight that is adapted by the training algorithm.

The logic of the neural network, which is similar to that of the brain, differs from the basic logic that is implemented in computers. However, neural networks can be implemented on a higher level in a computer. For a human to recognise a face takes only a fraction of a second, but a computer with the correct software will take much longer. Pattern recognition, which is only one of many abilities that the neural network tries to copy from the human brain, is extremely time-consuming when using normal Boolean logic. When the neural network is implemented on a computer, the advantages gained from solving highly non-linear problems make it a viable option, however.

3.2 Network topologies

Certain network architectures tend to be more efficient with certain training or learning algorithms. A few topologies are mentioned below together with a discussion of the training algorithms. These algorithms are implemented in the MATLAB[®] neural network toolbox, since MATLAB[®] was used in this study to develop neural networks to solve the various problems.

3.2.1 Multilayer feed-forward networks

The MATLAB[®] function to create a feed-forward network, either single or multilayer, is *newff*. The application of this function will be explained in the following experiments. These networks implement neurons in layers (See figure 3.3). Each layer could have an arbitrary number of neurons. For more complex problems the number of neurons is increased to retain the *knowledge* that is needed. Inserting another layer of neurons may also enable the network to see more complex behaviour in its training data.

The number of times the training data is fed to the network, called the number of epochs, is also important to ensure that the network captures the behaviour of the data. There is a trade-off between the number of neurons, the number of layers and the time required to train the network. Furthermore, problems are sometimes better solved with a less complex network. Figure 3.3 illustrates the multilayer feed-forward network topology.

According to the MATLAB[®] helpfile on multilayer architectures, the multilayer neural network is quite powerful, since it states that a two-layer network can approximate any function to an acceptable degree of accuracy (Also refer to Haykin (1999:229)).

To illustrate how the multilayer network fits together, one can visualise the neuron of figure 3.2 in the place of each neuron in figure 3.3. The number of inputs and thus the weights depend on either the number of inputs for the first layer, or the number of neurons in the previous layer in the case of a hidden layer. The first layer is normally called the input layer and the following layers are called the hidden layers. The last layer is sometimes called the output layer.

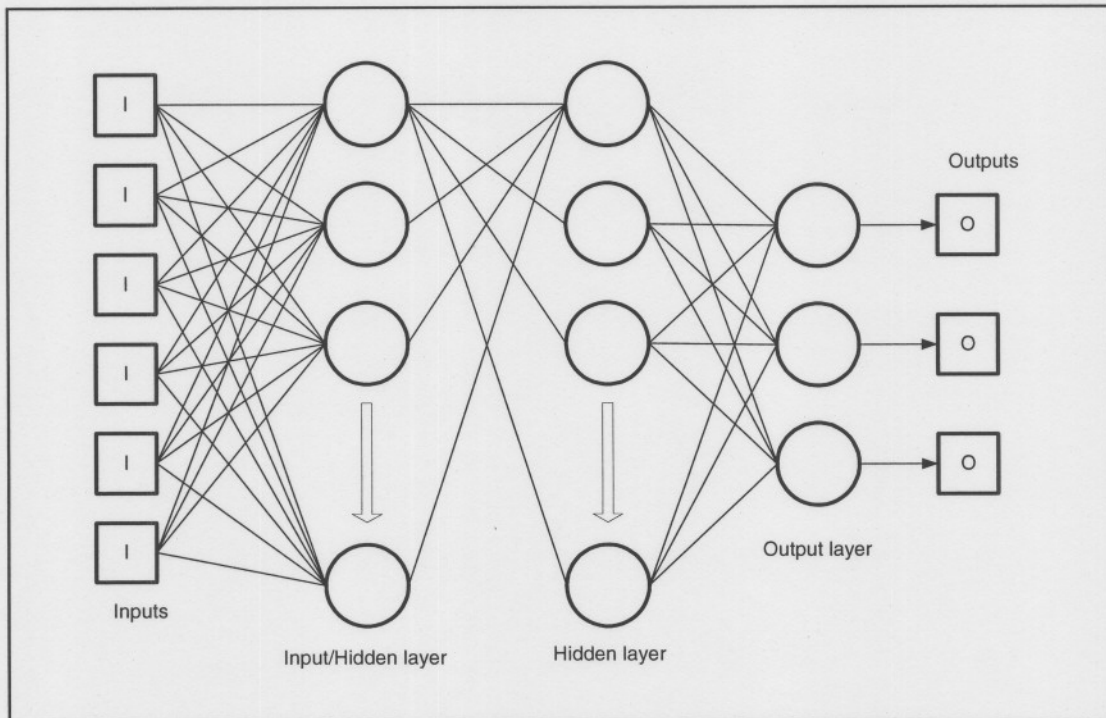


Figure 3.3: A multilayer feed-forward network architecture.

The connections in figure 3.3 between the first hidden layer and the second are not fully completed to keep the figure uncluttered. For the normal multilayer feed-forward network the neurons are fully interconnected.

Function	Description
Traingd	Gradient descent back-propagation
Traingdx	Gradient descent with momentum and adaptive learning rate back-propagation
Trainlm	Levenberg-Marquardt back-propagation
Trainrp	Resilient back-propagation
Trainbfg	BFGS quasi-Newton back-propagation

Table 3.1: Commonly used MATLAB[®] training functions.

Of all the available training algorithms in MATLAB[®], the more popular ones are listed in table 3.1. These are different kinds of back-propagation algorithms. Back-propagation training is a very common, though effective method. It is seen in most of the literature on neural networks.

The purpose of the activation function is explained with reference to figure 3.4. After the inputs have been multiplied with their respective weights and added together, the results are fed into this activation function. Non-linear functions like the log-sigmoid function, *logsig*, and the tan-sigmoid function, *tansig*, allow the network to map non-linear patterns. The linear function *purelin*, is not fit for mapping non-linear patterns.

NEURAL NETWORK THEORY

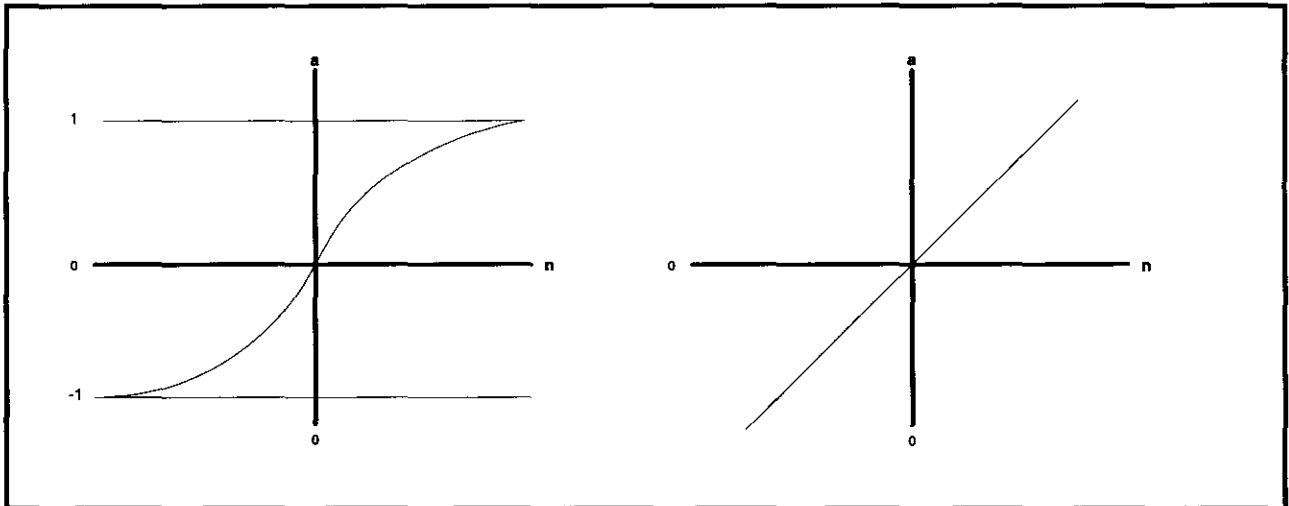


Figure 3.4: The tansig (left) and purelin (right) activation functions.

Throughout the experiments in this paper it was found that *trainrp* and *traingdx* were the best algorithms in terms of time and effectiveness. The *trainlm* algorithm was considered but as it takes much longer to train and uses much more memory, it was not used. MATLAB[®] also suggests that those algorithms listed should be applicable for multilayer networks.

Resilient back-propagation, *trainrp*, disposes of an inherent problem when using *logsig* activation functions in the network. The derivative between two large absolute values on the *n*-axis, figure 3.5, is very small. Algorithms using this derivative to adjust connection weights will thus take very long to train because this derivative is so small. Note that the sigmoid function allows only outputs between 0 and 1.

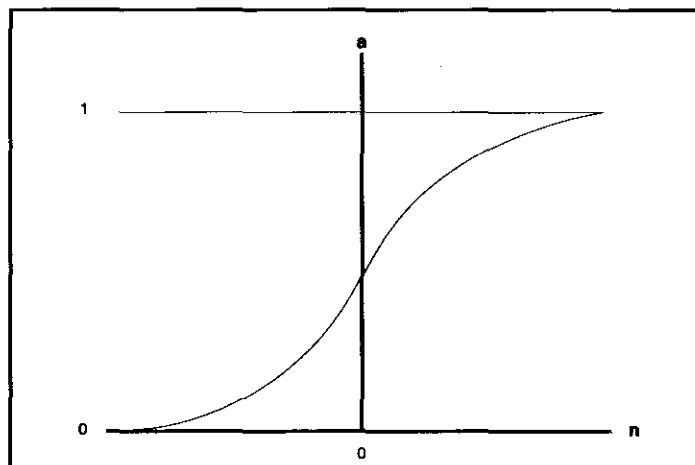


Figure 3.5: The log-sigmoid activation function.

NEURAL NETWORK THEORY

Resilient back-propagation uses the derivative only to obtain the sign of the derivative. This shows to which side the weight should be adjusted. When the derivatives at two successive points have the same sign, the weight is incremented by *delt_inc* and is decremented by *delt_dec* when they have different signs. Should the sign change every other derivative, the value *delt_dec* is decreased to prevent the algorithm from oscillating. These parameters are inherent to this training algorithm. Of course, when the derivative is zero, the weight is left unchanged. If the signs of a few successive derivatives were the same, *delt_inc* is increased to speed up the process. The network will stop training when:

- The number of *epochs*, set by programmer, is reached;
- the maximum amount of *time* has expired;
- the performance has reached the predetermined *goal*;
- the gradient of the performance is the predetermined level, *mingrad*;
- the “Stop Training” button is pressed (See figure 3.6).

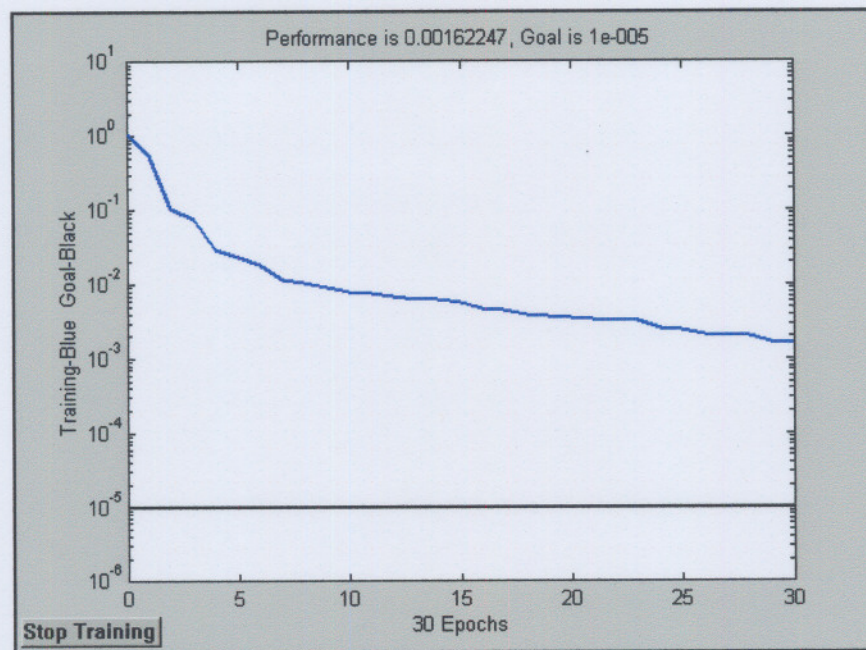


Figure 3.6: Information window while training a MATLAB® neural network.

In figure 3.6, the maximum number of epochs was set at 30. The performance *goal*, which is the mean-square-error between the network output and desired output, was set at 10^{-5} . When the gradient of the training performance is below *mingrad*, the algorithm knows that the performance curve will not reach the *goal*. That is at least within the set number of *epochs* or amount of *time* available. This applies to all the training algorithms.

NEURAL NETWORK THEORY

Gradient descent with momentum and adaptive learning rate back-propagation or *traingdx* is another training algorithm that is often applied in this paper. Were it not for the momentum and adaptive learning rate, gradient descent would not have been attractive. It is very slow when the learning rate is too small, while it will oscillate if the learning rate is too big. The learning rate is simply the step size factor by which the weights are changed.

The momentum *mc* in equation 3.2 (MATLAB® helpfile) is multiplied with the previous change of weights *dXprev*, the learning rate *lr*, and the derivative of the performance *dperf/dX*. The performance is with respect to the weights and the bias weights if implemented

$$dX = mc \times dX_{prev} + lr \times mc \times dperf/dX \quad (3.2)$$

The momentum increases the adaptation of the weights when the derivative of the error surface has the same sign for a number of successive training steps. Note that if the momentum *mc* is too big the algorithm might end up oscillating as well.

The learning rate *lr* is adapted during training. If the performance decreases toward the goal, the learning rate of the following epoch is increased by the factor *lr_inc*. The performance is determined by the generated output's deviancy from the output. If the performance increases by more than the parameter *max_perf_inc*, the learning rate is adjusted by the parameter *lr_dec* and the change, which increased the performance, is ignored. The parameters printed in italics are preset in MATLAB® to optimal values, although these could be edited by the programmer.

3.2.2 Recurrent networks

A recurrent neural network only differs from a normal feed-forward network in that it has feedback. Figure 3.7 illustrates the basic architecture of a recurrent network. The MATLAB® neural network toolbox has two types of recurrent network topologies, Elman and Hopfield networks, of which only the Elman network is considered. The Hopfield network is a self-organising map topology.

The Elman neural network in the MATLAB neural net toolbox has a basic structure of two layers, one hidden layer and an output layer. Because of the recurrent connection, this recurrent network is able to detect and generate time-varying patterns (MATLAB® helpfile on recurrent networks). It

NEURAL NETWORK THEORY

also recommends that when using the Elman network, one should use *traingdx* for the learning function and the *tansig-purelin* activation function combination for the two layers.

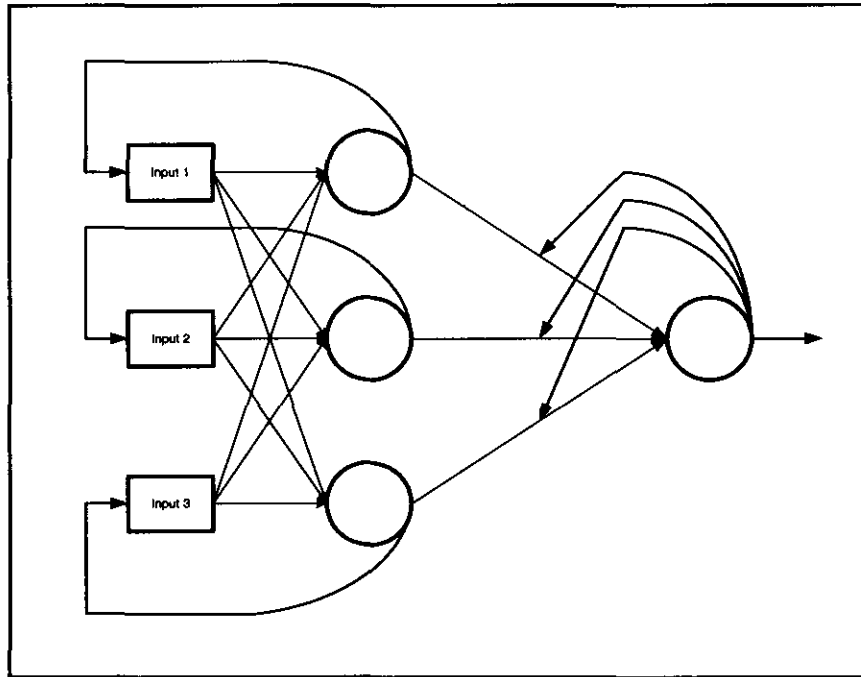


Figure 3.7: Basic recurrent network architecture.

4 RESIDUAL CLASSIFICATION

MATLAB SIMULINK[®] models are developed to model a plant from which residuals are generated. An investigation is done on normal, static, feed-forward neural networks for classifying these residuals. These networks are tested by adding noise to the residuals or quantising the residual data. These networks were developed on a single-input single-output and a multi-input single-output model.

RESIDUAL CLASSIFICATION

4.1 Problem discussion

The idea of this phase is to investigate and test a *classification* neural network. This network will classify residuals from the model in figure 4.1. A static neural network, which is a neural network without time delays, will be used for the classification.

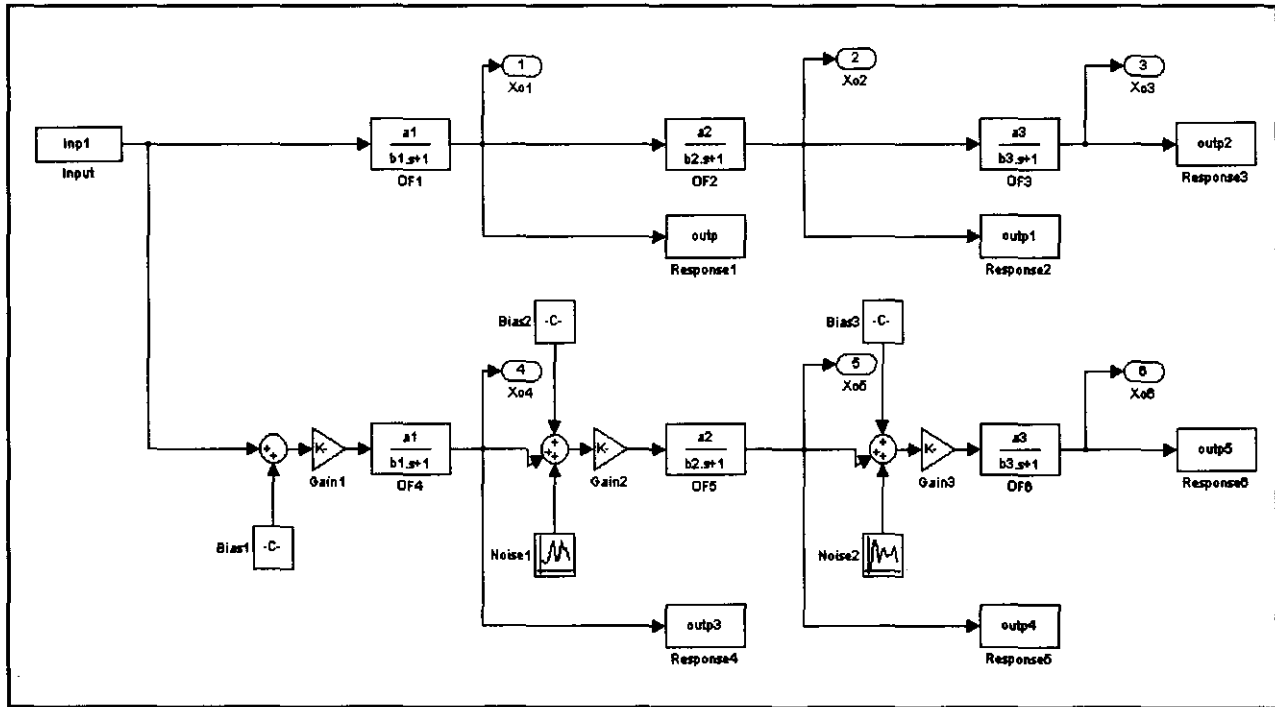


Figure 4.1: The parallel processes in SIMULINK®.

A SIMULINK® model consisting of two similar processes that are in parallel with each other is implemented. The only difference between these two processes is that in the lower process bias and gain errors can be inserted (Figure 4.1). The idea is to take measurements at corresponding points in these two systems. By subtracting these corresponding measurements one gets residuals that could be used to identify the fault that might be present in the process (Figure 4.2). These residuals can either be directly used or some statistical manipulation could be done beforehand. Conceptually, this FDD method could be implemented with the physical plant as the lower system and the reference model as the upper system.

The upper process in figure 4.1 models the system with no faults and can be replaced by a dynamic neural network that emulates the process. This dynamic neural network, which is a network with either feedback or time delays or both, is discussed in the next chapter.

RESIDUAL CLASSIFICATION

The process at the bottom represents the actual plant where system faults can artificially be inserted. Note that for FDD systems one may use any suitable model.

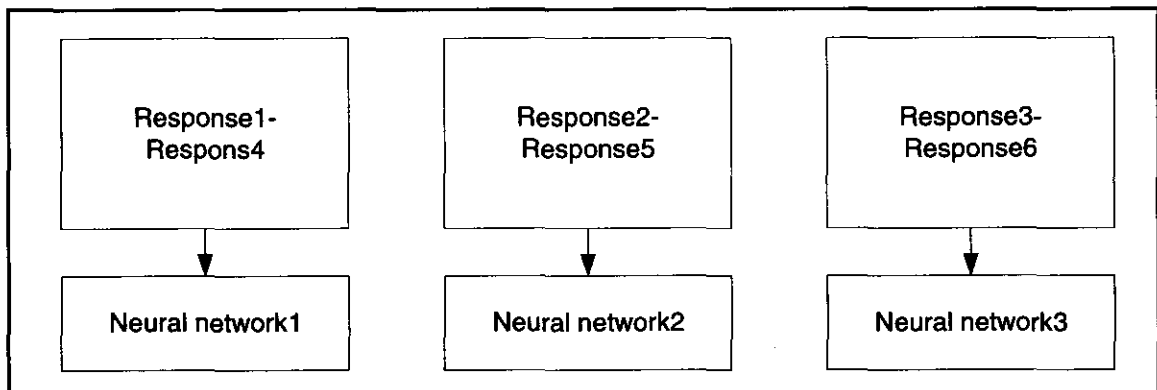


Figure 4.2: Input to the classification networks.

4.2 Detectability and isolability

Detectability is how easily a fault can be picked up or a significant residual can be generated. Isolability is how easily a fault can be distinguished from others. These concepts are important for the classification of residuals.

Detectability is highly dependent on the difference between the faulty and the non-faulty situation, while one can also look at the fault-to-noise ratio (Basseville, 1999). The isolability is again dependent on the information contained in the residual about that specific fault. When using statistical methods for an FDD system, these concepts play an important role in designing an effective system.

With neural networks, the detectability of the residuals depends on how accurately normal operation of the plant is modelled by the neural net. It follows that an inaccurate model may cause the difference between faulty and non-faulty operation to be either too small to detect or so large that no information about any fault may be obtained. The detrimental influence of noise on the residual will be considered in later sections.

Noise may cause the level of a residual to rise, creating a false indication of a fault. The sensitivity of the system depends on the set tolerance (Dummermuth, 1998). If the system tolerance is too high, many deviations might be overlooked, giving false negative indications of faults.

RESIDUAL CLASSIFICATION

On the other hand, if the tolerance or threshold were too low, many false indications of faults will be generated. The issue as (Dummermuth, 1998) stated is the tolerance of the system. This will be regarded in detail in chapter 6.

4.3 Methodology

4.3.1 SIMULINK®

The system in figure 4.1 is a model that is based on a fault detection system that could be implemented on an actual process. Sets of training data and testing data are obtained from this SIMULINK® model. A MATLAB® programme was developed that prepares the system for certain faults (or no faults) and then activates the SIMULINK® model. The faults and their target values are set in the programme. The residuals from the SIMULINK® model are saved in a MAT file together with their target values. These residuals are then used to train and validate the classification networks.

The design of the training sets involved some trial and error. A description of how the training sets were chosen will be shown briefly later on. The neural network programme loads the training data from the MAT file and trains with it. The SIMULINK® model is used to simulate a plant that could develop faults as well as a fault-free reference plant.

The SIMULINK® model is a third-order process, where each stage is represented by a transfer function. These transfer functions are all first-order functions like equation 4.1, and have variables, a and b , that can be changed to represent an internal failure of some kind.

$$G(s) = \frac{a}{sb + 1} \quad (4.1)$$

The reference model and test model enable one to insert faults at different locations. Gain and offset or bias faults are investigated since these are symptoms of a wide variety of plant and instrumentation faults. Zero drift in sensors, incorrect setting of stroke of control valves, fouling of heat exchangers and the effects of ambient temperature are examples. These faults are inserted in the lower part of the parallel system in figure 4.1. The input to the model is put in the workspace of MATLAB® from where it is passed to the models. The six outputs are also stored in the workspace by SIMULINK® and are thus passed on to MATLAB® where it can be manipulated.

RESIDUAL CLASSIFICATION

This system is a single-input single-output (SISO) system with all three states measured. To test the networks with a more complex situation, a multi-input single-output (MISO) system will be investigated in the latter stages of this section.

After the networks have proved that faults can be isolated in the ideal situation, classification of the residuals will also be tested when noise is added. For the more complex MISO system, the data will also be quantised to simulate quantising noise.

4.3.2 The data

A train of 20 pulses similar to the example in figure 4.3 is fed to the SIMULINK[®] model. These pulses vary in amplitude, duration and sign, the sign of each pulse being the negative of the previous pulse.

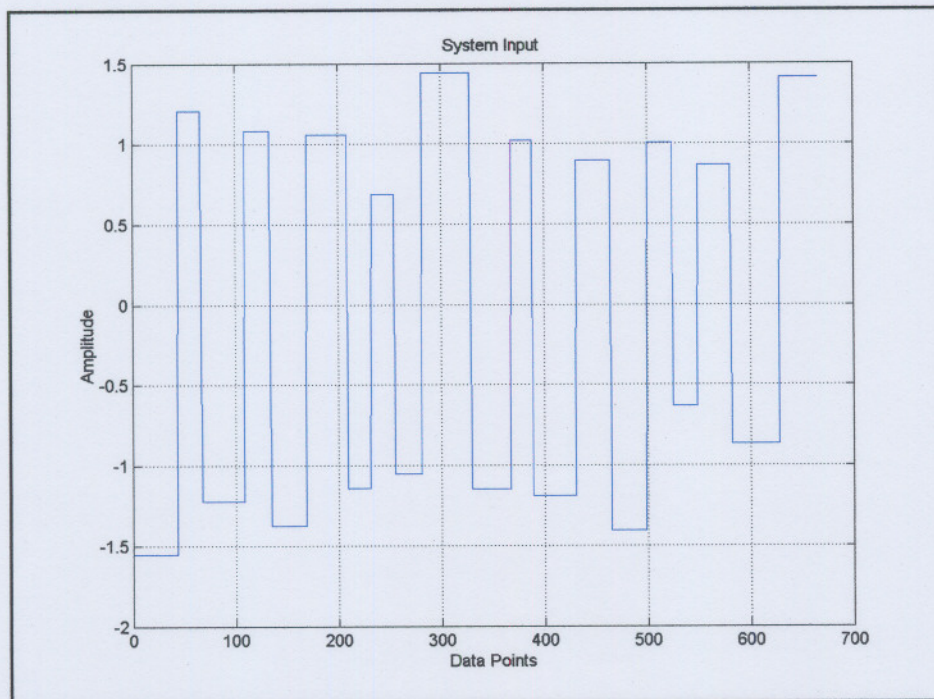


Figure 4.3: The input to the SIMULINK model.

This input represents the signal to an actuator that is used to control the input of the system represented by the SIMULINK[®] model. The pulse duration and amplitude is partly random. The six outputs are all of the same length and the outputs of the lower and upper system may differ in amplitude or dc level. The residuals are generated from these differences by subtracting the corresponding outputs of the upper system from those of the lower system.

RESIDUAL CLASSIFICATION

At this stage there are three residuals. From these residuals the system can be monitored for faulty behaviour. Figure 4.4 illustrates these signals, firstly for the zero offset or bias fault and secondly for a gain fault. In this simulation the effect of noise has not yet been investigated, but will be addressed later.

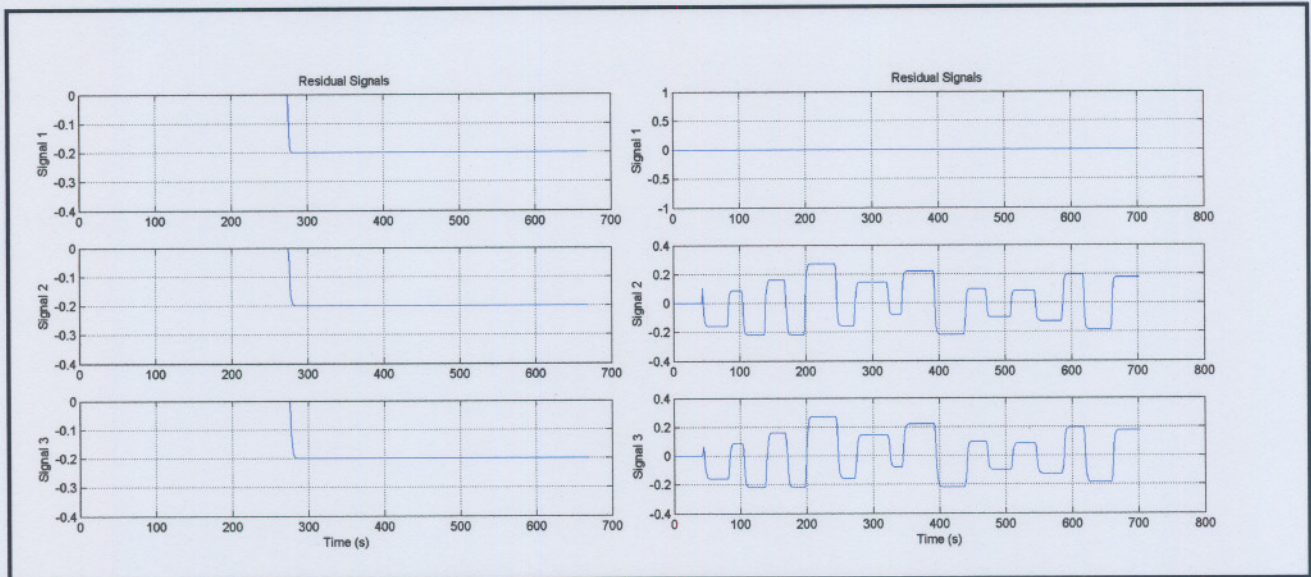


Figure 4.4: Signals representing a bias (left) and gain (right) fault.

When the programme detects a faulty situation the signal is stored in three fault windows. In this example, a total of 14 points taken every 5 steps are sampled and stored. If 14 consecutive points were taken, the fault windows might only contain the first transition of the gain fault causing it to look like a bias fault. The reason is that the first transition of the gain and the bias fault looks the same. Figure 4.4 shows the gain fault with its multiple transitions, while the bias fault window has only one transition.

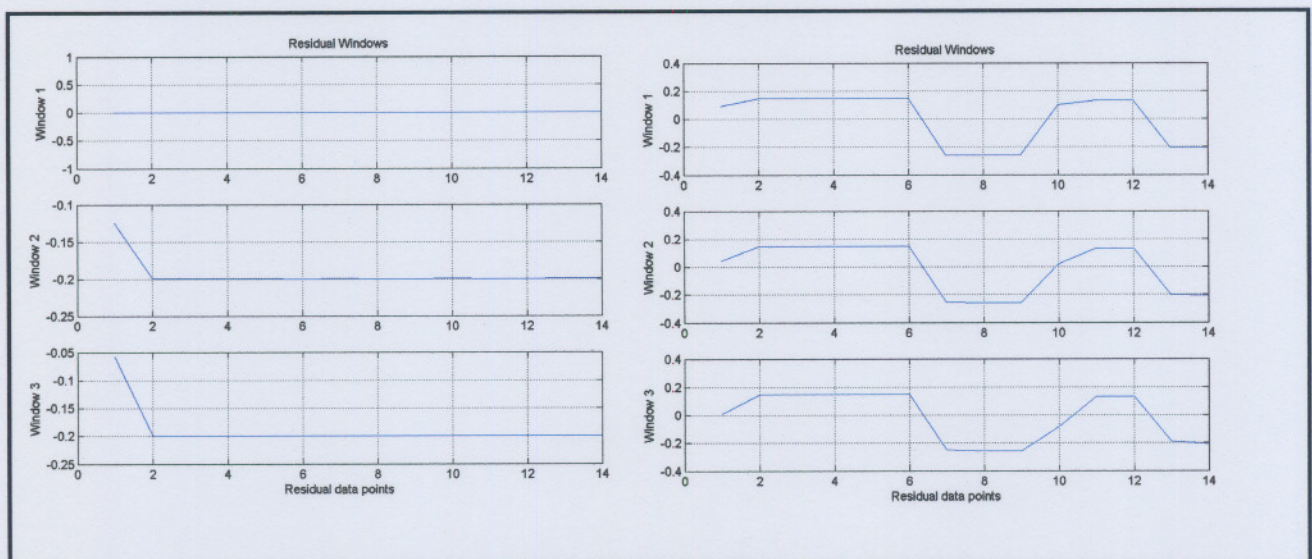


Figure 4.5: Fault windows for bias (left) and gain (right).

RESIDUAL CLASSIFICATION

Outputs for each residual window network			
No fault	1	0	0
Bias fault	0	1	0
Gain fault	0	0	1

Table 4.1: Possible outputs of the residual neural networks.

The inputs to the neural network consist of three windows similar to those in figure 4.5. In total there are 6 faulty conditions and then a faultless condition. Each window has three states, which is given in table 4.1. The fault status of the whole system depends on all three windows. For instance, there may be no fault in the first or even the second window, while the remaining window may contain a fault.

For the first scenario where there is a neural network for each window, the target values of the training data take the form of table 4.1. The output targets where one network is implemented for all three fault windows are illustrated in table 4.2.

Faults	Fault window 1	Fault window 2	Fault window 3	Target example (fault 3)
Fault 1(no fault)	No fault	No fault	No fault	0
Fault 2	Bias	Bias	Bias	0
Fault 3	No fault	Bias	Bias	1
Fault 4	No fault	No fault	Bias	0
Fault 5	Gain	Gain	Gain	0
Fault 6	No fault	Gain	Gain	0
Fault 7	No fault	No fault	Gain	0

Table 4.2: The output target format for the single-network scenario.

The faults in figure 4.4 are Fault 3 (left) and Fault 5 (right) respectively. To change the data in table 2 into the form of table 4.2, a conversion programme is implemented. The topology of these neural networks is discussed in the following section.

4.3.3 Neural network architectures

Two possible topologies are investigated. This investigation determines the fastest, most stable and most accurate topology in terms of training. The results of these investigations are given in the next section.

RESIDUAL CLASSIFICATION

4.3.3.1 Three-network topology

Each of the three networks is equivalent in terms of the number of inputs, neurons, layers and outputs. Each network is faced with unique data generated in different parts of the process. Although in this instance the data may look similar, it will not be trained with exactly the same data and end up with exactly the same weight matrix. Each network is set up in MATLAB® as illustrated below.

```
net=newff(z,[6 3],{'tansig','purelin'},'traingdx');  
net.trainParam.lr = 0.05;  
net.trainParam.epochs = 10;  
net.trainParam.goal = 1e-5;
```

A *for* loop is implemented to change the input/output training pair. This loop is inside another *for* loop that determines how many times the network will be trained with the whole training set. In this instance the number of epochs or the amount of times the network will train the specific input/output training pair is 10. If the outside loop runs 30 times, each training pair is trained 300 times.

4.3.3.2 Single-network topology

Here the network has 7 outputs instead of three due to the change in target format. A two-layer, 15-7 combination is illustrated as an example.

```
net1=newff(z,[15 7],{'tansig','purelin'},'traingdx');  
net1.trainParam.lr = 0.05;  
net1.trainParam.epochs = 50;  
net1.trainParam.goal = 1e-5;
```

The input to this network is the three fault windows appended, one to the other, which gives a 42-point input instead of three 14-point inputs. If one divided the input neurons by three, each fault window would be *assigned* 5 neurons. However, each input is connected to all the input neurons - 15 in this case - and thus the previous statement is not valid.

4.4 Results

4.4.1 Three-network topology

For all the combinations in the three-network experiments the deviation from the desired output is small. The issue of the investigation amounts to which network architecture is the most efficient in terms of speed. The best of these results will then be compared to the single-network topology.

RESIDUAL CLASSIFICATION

The time that the network needs to train depends on the complexity of the architecture. There is a trade-off between having a network with a very high complexity. To illustrate, consider two network architectures: one is a two-layer 3-3 combination and the other is a 30-3 combination.

View the simple network as a *dumb* child and view the complex network as an *intelligent* child. The *dumb* child will take much longer to achieve the *goal* than the *intelligent* child does. When the simple network is given 300 epochs with which to train, it might never pass the performance *goal* and then only the time limit, performance gradient or epoch-limit, would stop the training. The complex network might learn the solution much faster and only need 150 epochs before crossing the performance goal. The trade-off is when the network becomes more complex and the computation time for each epoch increases. Another issue is whether or not more complex networks solve the problem better.

An experimental run in which six different architectures were investigated was made. Each experiment was run 10 times. The measures by which the networks were compared are time in seconds and a cumulative error. The mean-squared-error of test samples are added together to generate the cumulative error. The test data is a set of 14 samples that comprises two sets of the possible 7 faults that could occur.

Architecture	3-3(1)	6-3(2)	9-3(3)	14-3(4)	18-3(5)	24-3(6)
Time (s)	1188.8	894.5	845.3	748.2	699.1	635.4
Cum. Error	1.32554	0.10524	0.14262	0.22883	0.26836	0.35748

Table 4.3: Performance comparison of the three-network topology.

As illustrated in figure 4.6, the speed by which the network train improves as the input neurons are increased. However, the accuracy decreases as the number of input neurons are increased (Figure 4.7). The number of epochs was 300 in each experiment.

The network that gave the best accuracy was compared with the network that took the least amount of time to train. The number of epochs was set at 1050 epochs. The idea was to see if the tendency remained when the networks were given more epochs. After 8 runs the answer was obvious. Figure 4.8 illustrates this clearly.

RESIDUAL CLASSIFICATION

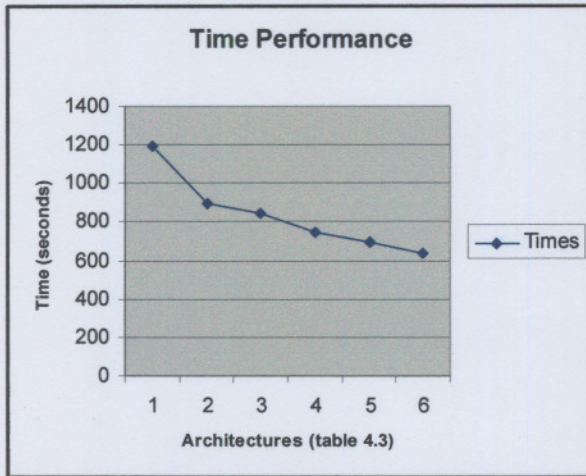


Figure 4.6: Time performance of the different topologies.

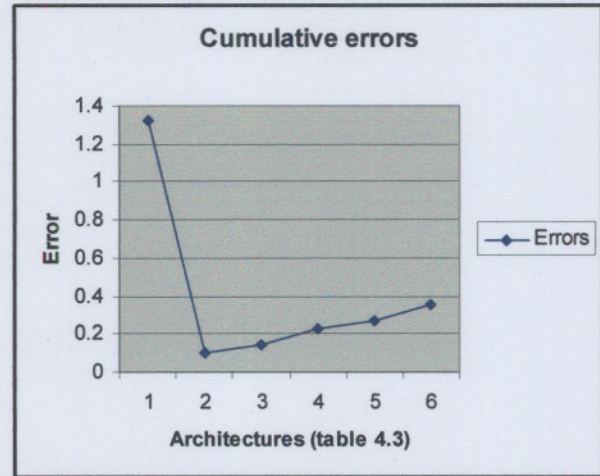


Figure 4.7: Accuracy of the different topologies.

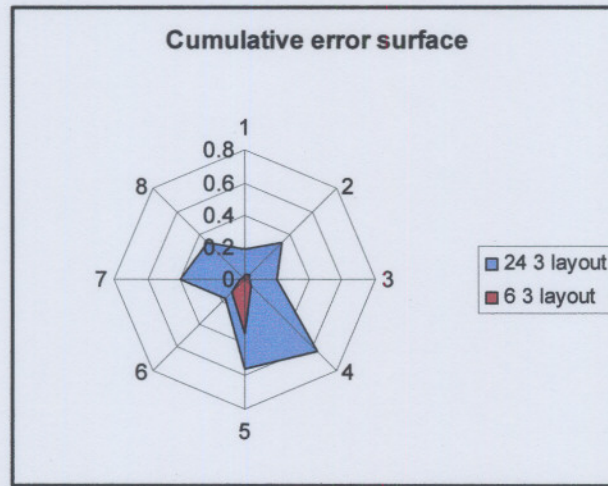


Figure 4.8: A radar plot of the cumulative errors over 8 runs.

Even though the 6-3 architecture took some 300 seconds (5 min) longer than the 24-3 network, the average cumulative errors over the 8 runs of this architecture is visibly much better. Numerically, it reads as follows: the 6-3 network has an error of 0.08246 and the 24-3 network has an error of 0.3469. Figure 4.8 also illustrates the consistency of the smaller network except for the one outlier at sample 5. The more complex network has less consistency if one looks at the angular surface in figure 4.8 (the ideal consistent surface would be a perfect octagon in this case).

Why is the complex network that trains 5 minutes faster the least accurate network? The reason might be *overtraining*. *Overtraining* is when the network trains the data too exact. If an input that varies just a little from its training data were given, the network cannot generalise and gives an inaccurate answer. Why would the larger network *overtrain* and not the smaller network?

RESIDUAL CLASSIFICATION

The simple network has 84 input weights and the complex network has 336 input weights. The complex network has almost twice the number of neurons than it has inputs, while the simple network has less than half the number of neurons than it has inputs. These characteristics logically allow the complex network to see the input in much more detail. It also has four times more input weights than the simple network to store the characteristics of the input. This might allow the larger network to learn the training data too exactly and might decrease its generalisation.

The test data comprises two sets of the 7 possible errors and it was obvious that the complex network struggled with the second set of gain errors. It was less obvious in the case of the simple network. This would mean that the second gain error set is different in some way to those in the training data. The simple network classified this variant much better than the complex network did. The conclusion is made that the complex network *overtrained* because of its *resources* and did not have the same generalisation ability as the simple network had.

4.4.2 Single-network topology

Will the single network perform better than the three-network topology? A single window of 14 points is less complex as an input than three windows together as an input. The network becomes more complex in the single-network scenario even if the total number of input neurons were less. The number of input weights in the three-network topology is, for 6 neurons in each network, $14 \times 6 \times 3 = 252$ input weights for all 3 networks together. For the single network the number of input weights for 15 input neurons is $15 \times 42 = 630$. The single-network topology is a more complex problem.

The *single-network* scenario was tested with architectures of two and three layers. A cumulative error and time, in seconds, were again taken as performance measures. The number of epochs was kept at 1500, while the number of neurons and layers were varied. The trade-off between accuracy and speed was investigated by varying the complexity of the architecture. Detailed results are given in table A.2 and A.3 in the appendix.

RESIDUAL CLASSIFICATION

Architecture	7 7 7(1)	9 7 7(2)	12 7 7(3)	9 9 7(1)	12 9 7(2)	15 9 7(3)
Time (s)	2163.1	2170.3	2136	1824.2	1986.5	2008.5
Cum. Error	1.11928	1.00877	0.702375	0.65277	0.77254	0.68951
Architecture	7 12 7(1)	9 12 7(2)	12 12 7(3)	15 12 7(4)	7 15 7(1)	9 15 7(2)
Time (s)	1606	1626	1729.6	1719.5	1449.9	1511.3
Cum. Error	0.82208	0.61401	0.61811	0.50778	0.6766	0.60638
Architecture	12 15 7(3)	15 15 7(4)	9 18 7(1)	12 18 7(2)	15 18 7(3)	
Time (s)	1545.2	1528.5	1377.9	1393.7	1471.7	
Cum. Error	0.64614	0.61474	0.85687	0.56688	0.75755	

Table 4.4: Performance comparisons of the single-network topology.

When three layers are used there are more combinations as illustrated in table 4.4. These values are averages taken over 10 experiments. To simplify the comparison of the different architectures, the architectures were grouped according to the number of neurons in the middle layer. In figures 4.9 and 4.10 they are plotted as the neurons in the input layer increases from architecture 1 to 3 or 1 to 4 (See table 4.4).

Figure 4.10 shows that the more complex network architectures do not necessarily provide the best solution, although in terms of time performance the larger networks trained faster on average. Architectures that are less or more complex had inferior performances and were left out. With architecture 1 in figure 4.10, the network with 9 neurons in the middle layer performed the best. When the input neurons were increased, the performance of the network with 12 neurons in the middle layer outperforms the rest.

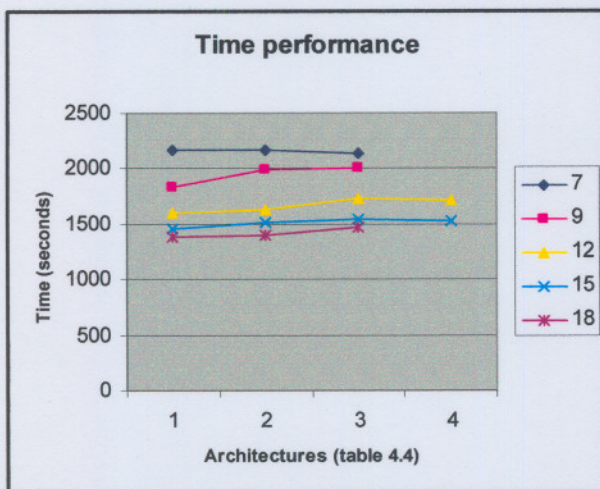


Figure 4.9: Time average plots of different topology groups.

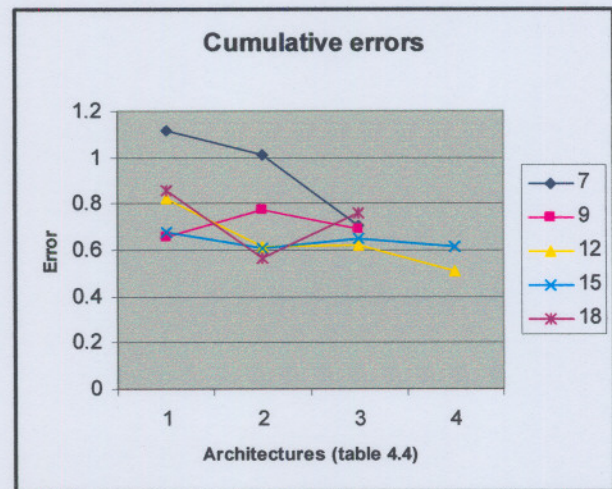


Figure 4.10: Cumulative error of the different topology groups.

RESIDUAL CLASSIFICATION

4.4.3 Multi-input single-output system

The MISO system is investigated to test whether residual classification using neural networks is efficient in a more complex environment. The process residuals up to now have been too simple to test the classification performance with quantised data. The schematic illustration of the model is given in the appendix, figure A.1. The first input is the one used in the SISO model and the second input is a constant block-wave.

The three residuals in this instance differ from each other and not only from those in the previous section. Two of them have three possible states and the third has five possible states. The final residual may contain (1) no residual, (2) a bias residual, (3) a constant block-wave that has 1 or -1 amplitudes, (4) a non-constant block wave with varying amplitude and (5) a block wave that has zero regions in-between a non-constant block-wave. Examples of the last three possibilities are illustrated in figure 4.11.

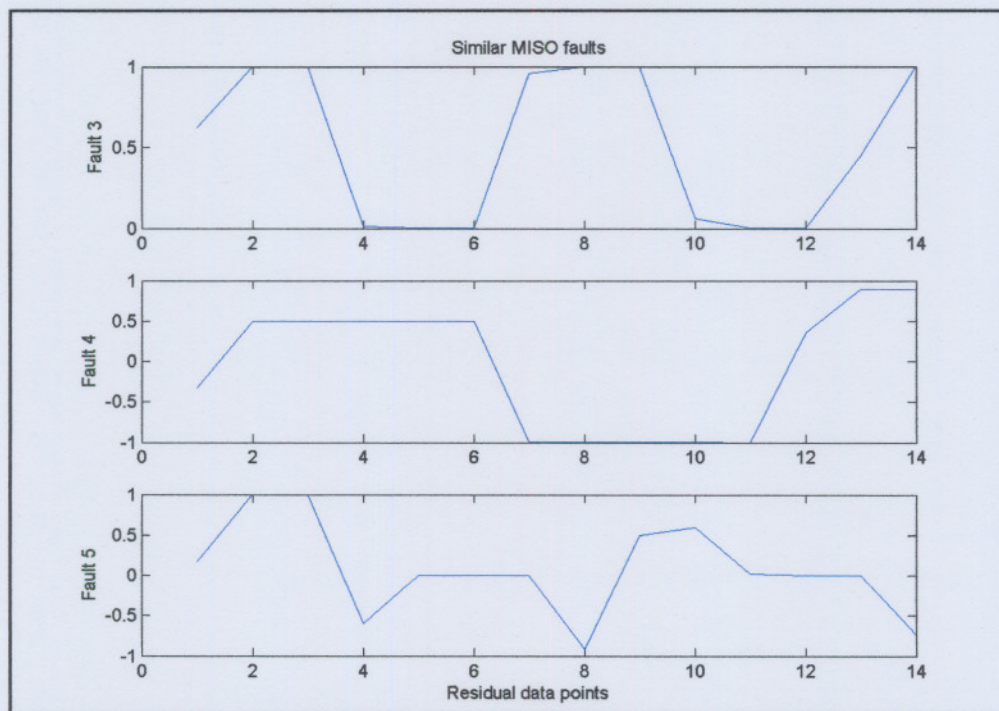


Figure 4.11: Three similar possibilities that increase classification complexity.

The similarity between these residuals causes uncertainty with regard to their classification. There were no misclassifications but the neural network struggled to generate definite or certain classifications in some instances. An example of such uncertainty is shown in figure 4.12.

RESIDUAL CLASSIFICATION

The “combined NN outputs” axis in figure 4.12 has 11 possible positions. This is the possibilities of all three classification networks one after the other. The composition is 3-3-6, which represents three faults for the first two classifications and 6 faults for the third classification network. The “combined test data” axis represents the three inputs, one after the other, which gives 42 data points in total. The figure layout is chosen to emphasise the uncertainty.

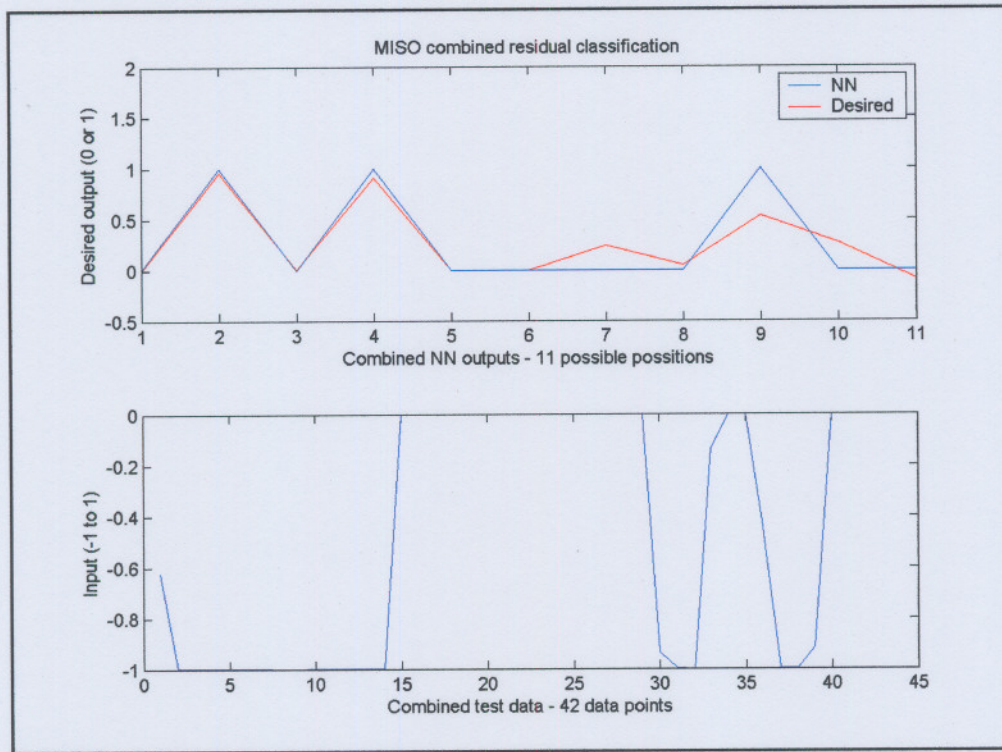


Figure 4.12: Neural network uncertainty with similar residuals.

4.4.4 Noisy data

In practice, the data that reaches the classification network will have some degree of noise present. This section takes the best network topologies from the previous section and adds noise to the testing data. The *three-network* topology's best *performer* is investigated. The 6-3 network was trained with noiseless data. Then noise was added to the testing data and the network was simulated with this noisy data. The noise added is Gaussian noise that was generated with the *randn* function.

The network was trained a few times and simulated with noiseless data. The most accurate network was saved and the noise was added. The percentage of the noise to the signal was taken firstly at ± 36 dB SNR and taken up to ± 13 dB SNR. The actual data is tabulated in the appendix,

RESIDUAL CLASSIFICATION

table A.4. Down to the 30 dB SNR mark there were no real deviation and no misclassifications. When the SNR reached the 24 dB mark, the network failed with some faults.

These misclassifications depended on where the larger noise components appeared. In some cases they appeared at places so that the network error was as low as 0.0289, which is just above the mean of the 30 dB SNR errors. In certain instances the cumulative error was large but there were no misclassifications. This is when the errors were spread over the whole test sample.

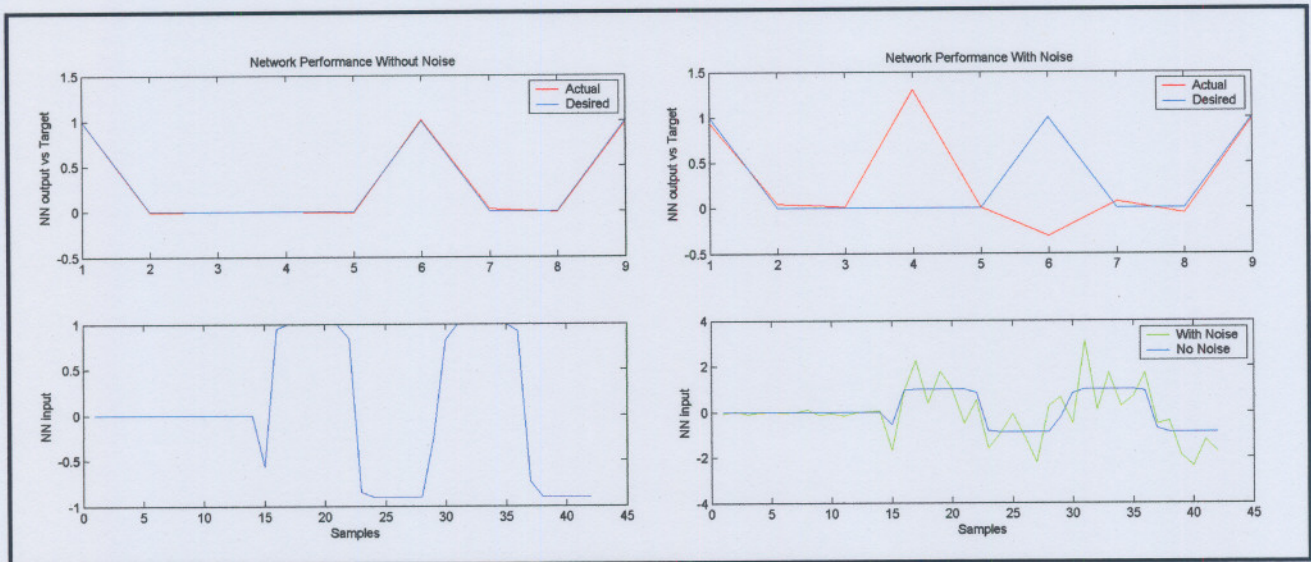


Figure 4.13: The same network with no noise and 13 dB SNR noise.

For the SISO process the input data is relatively simple and noise did not have a decisive effect on the output. The question might be whether the network performs with noise when the classification is not that clear-cut.

4.4.5 Quantised data

The effect that quantising of the input data has on the output of a neural network is investigated in this section. The MISO model residuals are used in this instance. The third neural network with the 5-output architecture is used. The five classes of residuals used for this network are sufficiently representative of the residuals that are used for the first two neural networks.

	No quant.	12-bit	8-bit	6-bit	4-bit	2-bit
Cum. MSE	0.2839	0.2839	0.2867	0.2979	0.4988	2.1432

Table 4.5: Cumulative MSE for quantised test set.

RESIDUAL CLASSIFICATION

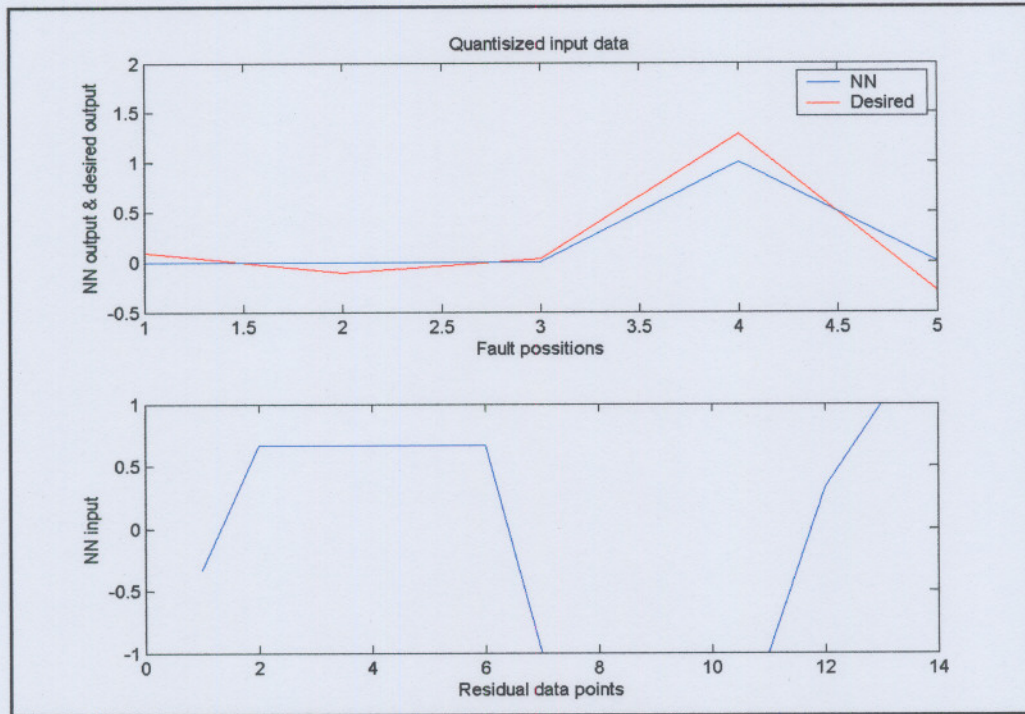


Figure 4.14: 2-bit quantized data.

The classification neural network was tested with 2-, 4-, 6-, 8- and 12-bit quantised input data. Table 4.5 numerically illustrates the effect of this quantising. The error that is present with no quantising is from the uncertainty discussed in section 4.4.4. The 12-, 8- and 6-bit quantising adds little to the existing error. The 4-bit quantizing almost doubles the existing error in this instance.

The 2-bit quantising does give a large error but some of the outputs were correctly classified in figure 4.14. The 2-bit case was the only experiment that gave misclassifications. The 4-bit experiment had one sample where the network gave fault 3 and fault 4 equal *likelihood*, when it should only have been fault 3.

4.4 Conclusion

The classification of the SISO model residuals posed no problem for the *three-network* topology. Even when noise was added, the generalising ability of the neural network prevented misclassification. It is also evident from this investigation that the use of more complex neural network architectures is not the probable solution. They might train faster, but they are more susceptible to *overtraining* and so generate bigger errors.

RESIDUAL CLASSIFICATION

The neural networks did show little uncertainty with the residuals of the MISO model. Many experimental runs with different architectures only minimised this uncertainty to some test samples. More in-depth research is needed to locate the origin, which is beyond the scope of this paper. Finally, the neural network concept overcame the problem of classifying quantised data. In this instance, substantial problems only started below 4-bit quantising.

5 INDUCTION MACHINE BENCHMARK

The modelling ability of normal feed-forward neural networks with time delays is investigated. An induction machine model developed in SIMULINK® is implemented to obtain training data. The neural network's ability to model a dynamic non-linear process is investigated.

INDUCTION MACHINE BENCHMARK

5.1 Purpose

A practical benchmark is chosen to gain knowledge about the feasibility of the modelling concept investigated below. In this instance a simulation of an induction motor was chosen as the benchmark. Panaitescu (Panaitescu R.C) developed an induction motor simulation for the steady state case of a squirrel-cage induction machine. Modelled data will be generated to train a dynamic neural network with the dynamic behaviour of this induction motor. The question is whether the neural network will be able to simulate the induction machine?

5.2 Background

This mathematical model of an induction machine is implemented in SIMULINK®. A brief discussion of the workings of the induction motor and the mathematical equations that are of importance are given.

A squirrel-cage induction motor is the specific machine that is simulated. This machine does not have rotor voltages, which simplifies the problem. Figure 5.1 (<http://www.shef.ac.uk/materials/SSM/imi/squicage.jpg>) gives an illustration of the basic idea behind a squirrel-cage induction machine.

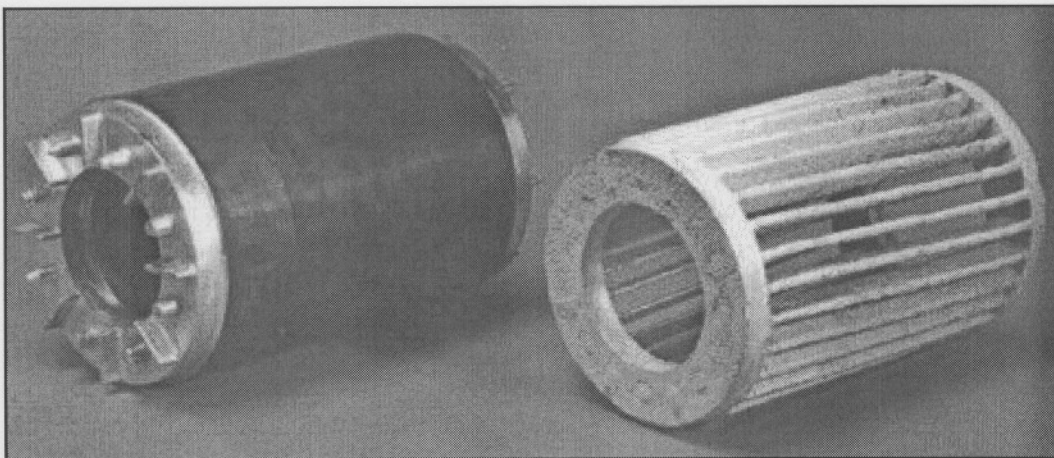


Figure 5.1: The rotor cage of a squirrel-cage induction machine.

The cage in figure 5.1 is commonly made of copper and is usually moulded within a less conductive metal. The magnetic fields produced by the 3-phase windings give rise to a current flow in the copper bars of the rotor. These currents in turn produce their own magnetic field that interacts with the stator field to produce an electromagnetic torque, which turns the rotor.

INDUCTION MACHINE BENCHMARK

This magnetic torque is the output target that the neural network will be trained with. The equations that describe the motor mathematically are given below their symbol definitions in table 5.1.

▪ Voltage equations:

$$v_{sd} = R_s i_{sd} - \omega_d \lambda_{sq} + \frac{d}{dt} \lambda_{sd} \quad (5.1)$$

$$v_{sq} = R_s i_{sq} + \omega_d \lambda_{sd} + \frac{d}{dt} \lambda_{sq} \quad (5.2)$$

$$v_{rd} = 0 = R_r i_{rd} - \omega_{dA} \lambda_{rq} + \frac{d}{dt} \lambda_{rd} \quad (5.3)$$

$$v_{rq} = 0 = R_r i_{rq} + \omega_{dA} \lambda_{rd} + \frac{d}{dt} \lambda_{rq} \quad (5.4)$$

▪ Flux linkages:

$$\lambda_{sd} = L_s i_{sd} + L_m i_{rd} \quad (5.5)$$

$$\lambda_{sq} = L_s i_{sq} + L_m i_{rq} \quad (5.6)$$

$$\lambda_{rd} = L_r i_{rd} + L_m i_{sd} \quad (5.7)$$

$$\lambda_{rq} = L_r i_{rq} + L_m i_{sq} \quad (5.8)$$

▪ Electromagnetic torque:

$$T_{em} = L_m (i_{sq} i_{rd} - i_{sd} i_{rq}) \quad (5.9)$$

▪ Electrodynamics

$$J_{eq} \frac{d}{dt} \omega_m = T_{em} - T_L \quad (5.10)$$

$$\frac{d}{dt} \theta_m = \omega_m \quad (5.11)$$

$$\text{or } \left(\frac{d}{dt} \theta_d = \omega_d; \frac{d}{dt} \theta_{dA} = \omega_{dA} \right).$$

INDUCTION MACHINE BENCHMARK

Name	Description	Units
v_a, v_b, v_c	Stator phase-voltages	V
i_a, i_b, i_c	Stator phase-currents	A
$\vec{v}_s(t)$	Stator voltage space-vector	V
$\vec{i}_s(t), \vec{i}_r(t)$	Stator and rotor current space-vectors	A
v_{sd}, v_{sq}	Stator voltage vector components in rotational reference frame	V
$v_{s\alpha}, v_{s\beta}$	Stator voltage vector components in stator oriented frame	V
i_{sd}, i_{sq}	Stator current vector components in rotational reference frame	A
$i_{s\alpha}, i_{s\beta}$	Stator current vector components in stator oriented frame	A
i_{rd}, i_{rq}	Rotor current vector components in rotational reference frame	A
$\vec{\lambda}_s(t), \vec{\lambda}_r(t)$	Stator and rotor flux space-vectors	Wb
$\lambda_{sd}, \lambda_{sq}$	Stator flux vector components in rotational reference frame	Wb
$\lambda_{rd}, \lambda_{rq}$	Rotor flux vector components in rotational reference frame	Wb
ω_d	Synchronous angular speed	rad
ω_{dA}	Slip angular speed	rad
ω_m	Rotor (mechanical) angular speed	rad
$\theta_d, \theta_{dA}, \theta_m$	Angles in rad, corresponding to the above speed types	-

Table 5.1: Symbol definitions.

Name	Description	Units
P_n	Motor Rated Power	W
I_n	Rated Phase Current (rms value)	A
V_n	Rated Phase Voltage (rms value)	V
P	Number of pole-pairs	-
f_n	Rated Frequency	Hz
T_n	Rated Torque	Nm
T_L	Load Torque	Nm
J_{eq}	Moment of Inertia (equivalent)	Kg m ²
R_s	Stator Resistance	Ω
R_r	Rotor Resistance	Ω
L_m	Magnetizing Inductance	H
L_{ls}	Stator Leakage Inductance	H
L_{lr}	Rotor Leakage Inductance	H
$L_s = L_{ls} + L_m$	Stator Equivalent Inductance	H
$L_r = L_{lr} + L_m$	Rotor Equivalent Inductance	H

Table 5.2: Induction machine parameter definitions.

5.3 Methodology

5.3.1 Simulation

Clark and Park transforms are implemented to simplify the process. These transforms implement another reference frame that rotates with the rotor or the magnetic field. The equations defining the process are transformed to the state space, which allows for easy use in MATLAB®/SIMULINK®. The parameters in these equations are defined in table 5.1. The equation

$$\dot{\mathbf{i}} = \mathbf{A}^{-1} \cdot \underline{\Lambda} \quad (5.12)$$

is obtained, which gives the current matrix. $\mathbf{A}$ is a time-invariant matrix that is given by

$$\mathbf{A} = \begin{bmatrix} L_s & 0 & L_m & 0 \\ 0 & L_s & 0 & L_m \\ L_m & 0 & L_r & 0 \\ 0 & L_m & 0 & L_r \end{bmatrix} \quad (5.13)$$

and Λ is

$$\Lambda = \begin{bmatrix} \lambda_{sd} \\ \lambda_{sq} \\ \lambda_{rd} \\ \lambda_{rq} \end{bmatrix}, \quad (5.14)$$

which is the flux linkage components in the d - q reference frame. This d - q reference frame is due to the Clark and Park transform. $\mathbf{A}^{-1}$ is only dependent on the motor's parameters and can thus be computed before the simulation and used within. The subscripts indicate the d - q reference frames for the stator, s , and the rotor r .

The electromagnetic torque output of the simulation, which is implemented in training the neural net, is given by

$$T_{em} = L_m (i_{sq} i_{rd} - i_{sd} i_{rq}) \quad (5.15)$$

The simulation model is shown in figure 5.2. For details on deriving the model and implementing it in SIMULINK®, refer to Chee-Mun Ong (1998).

INDUCTION MACHINE BENCHMARK

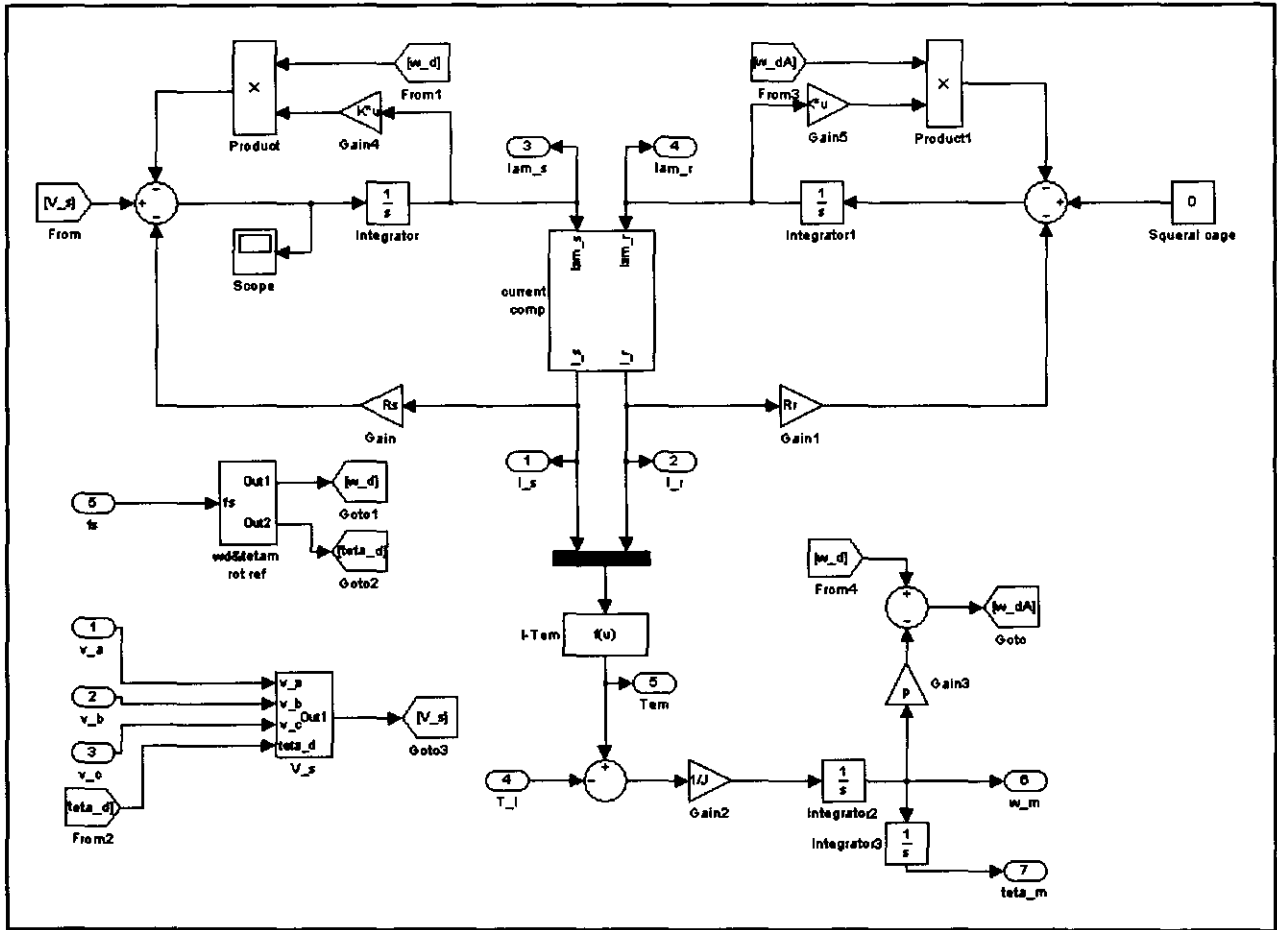


Figure 5.2: A SIMULINK® model of the induction machine.

A three-phase voltage at constant frequency is taken as an input for the simulation model. The model input that is used for training the neural network is the mechanical load. The system outputs, I_s , I_r , λ_s , λ_r , ω_m , and θ_m are not used for training in this instance. T_{em} , the electromagnetic torque is chosen arbitrarily as the output, and the mechanical load as input.

5.3.2 Training data

The performance of a neural network is as dependent on a representative set of training data as on the architecture. A smaller training set is initially used to get an idea of what architectures would be suitable. This is done with the assumption that an architecture that performs best with a smaller training set would also perform best with a larger training set. Further motivation for initially using a smaller training set is given in section 5.3.3.

INDUCTION MACHINE BENCHMARK

The training data is made up of the normal step input and response. Multiple step inputs in the form of a block waves and their responses are used to capture a wide bandwidth of the model's character. In practice, a motor driving a generator that has a load, which is switched on and of, can represent this kind of input. An illustration of an input and output pair is shown in figure 5.3.

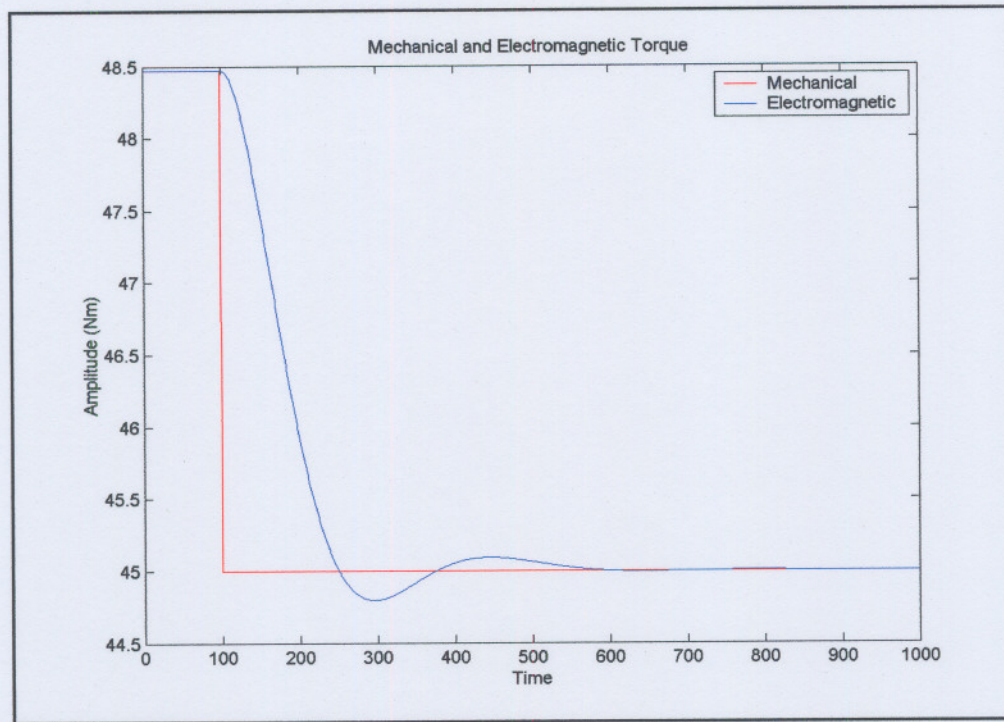


Figure 5.3: An input/output pair sample of the training data.

The data is normalised for the training set that needs to lie within a certain range, depending on how the neural network is defined. The sample in figure 5.3 is not normalised to show the steady state value of the input from which the step is made. This value is approximately 48.5 Nm and is consistent throughout the training set.

With this model the electromagnetic torque has negative values if the final step value were made too low. This means that the induction machine acts as an induction generator for that moment. In this instance the training data is monitored so that the dynamics of the induction machine is for an induction motor only.

Each sample that is generated by the SIMULINK[®] induction machine model has 1000 data points. The number of delays in the input weights would be too high and training would take very long. The training data is made up of every 5th point of the original data sample, which gives 200 data points.

5.3.3 Neural network topology

Dynamic neural networks have time delays, which gives the neural network a *short-term* memory. This memory is needed to keep track of what happened to the input a few moments before the current time. This characteristic is a prerequisite when working with time-dependent dynamic processes.

MATLAB[®] has two ways of giving a neural network memory. The first is with a recurrent neural network (See figure 3.7). The *newelm* function defines a recurrent neural network. Secondly a normal feed-forward network can be created with delays. These delays create a time memory line parallel with the neurons in each layer. Figure 5.4 illustrates how this concept is realized. This network has one input and 6 time delays. To keep the figure simple, all the connections are not shown.

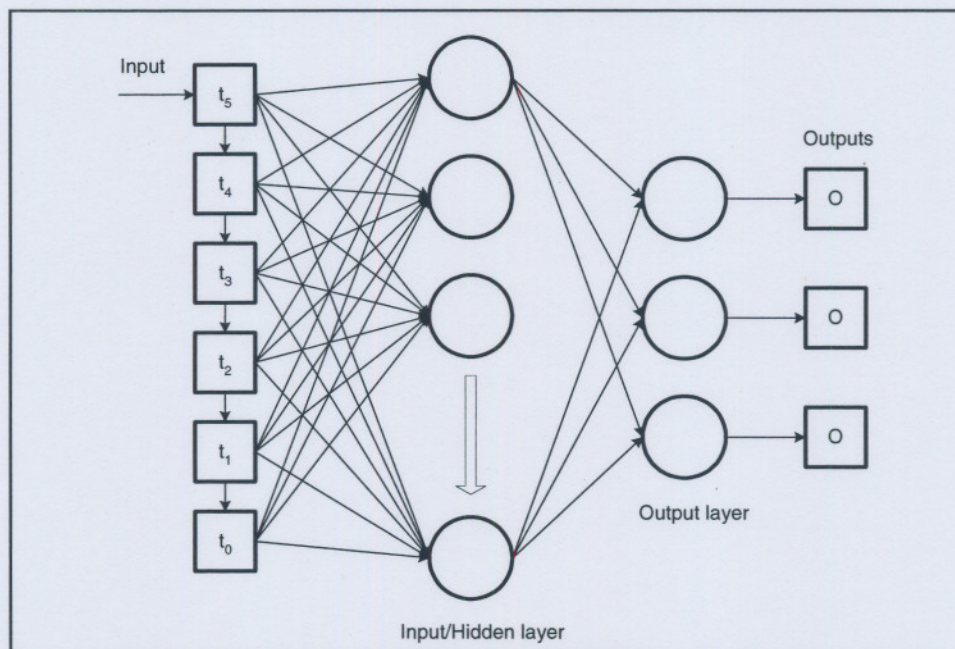


Figure 5.4: A feed-forward neural network with time delays [input at $t_0, t_1, \dots$].

A feed-forward neural network is defined by *newff* in MATLAB[®]. Both these functions are implemented as shown in section 4.3.3.1. The difference is that delays are inserted in the weights as illustrated below.

```
netind.inputWeights{1,1}.delays=linspace(1,20,20);
netind.layerWeights{2,1}.delays=linspace(1,20,20);
```


INDUCTION MACHINE BENCHMARK

In this example the input and layer weights are given 20 time delays. Delays in the hidden layers of a neural network lead to much more computational time, which means a very slow training network. The size of the training data set adds to computation time and this is why the initial training data set is kept effectively small.

5.4 Results

The *newelm* networks did not perform satisfactorily and no results are given. The *newff* network performed the best and is investigated further. The two-layer networks did achieve success with some test samples, but the networks mostly gave outputs with high frequency noise. The focus of this section will be on the three-layer neural network that has *logsig* activation functions in the first two layers and a *purelin* function in the output layer. The *traingdx* was the only training algorithm that produced useful outputs. The parameters that are varied to achieve optimal simulation results are the neurons in the input and hidden layers, the learning rate parameter, the number of epochs and the number of delays in each layer.

The main issue was finding a neural network that could *capture* the transfer dynamics of the process in question. If the process is given a step input and gives a response with some overshoot or ripple, would the neural network be able to simulate the ripple correctly? If the step were not *visible* to the network anymore, the network must rely on the dynamic information left in the time delayed weights. As figure 5.5 illustrates, this is not possible with insufficient delays. The orange and green rectangles represent the visual scope of the network. The orange window is taken just after the step input is out of *vision*, while the green window after the step response has *settled*.

The problem is clear. The network has exactly the same input, mechanical torque, for both windows but the output, electromagnetic torque, is totally different. The visual scope of the network is determined by the number of time delays. Figure 5.6 shows the response of a network that did not have sufficient time delays. The output goes to the settled value of the response.

INDUCTION MACHINE BENCHMARK

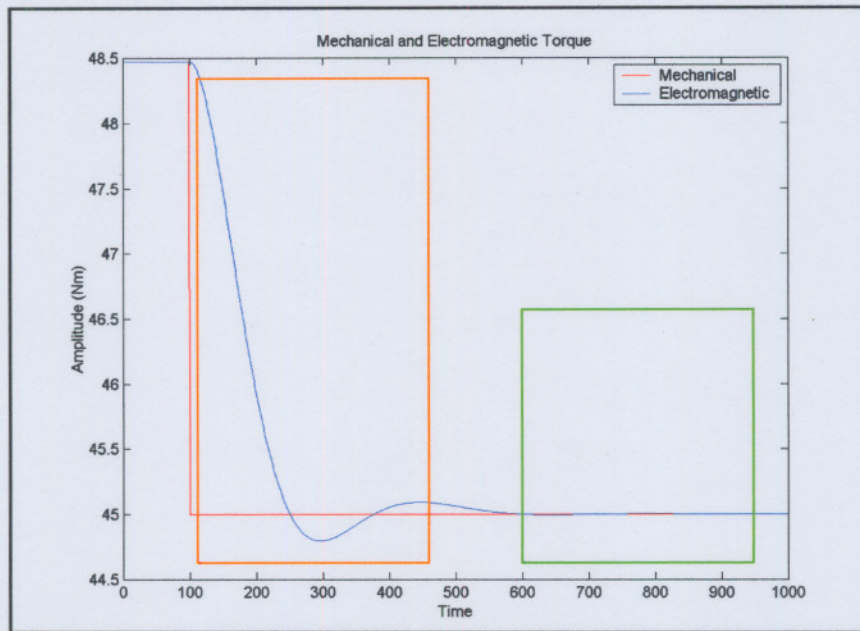


Figure 5.5: Same inputs (red) with different outputs (blue).

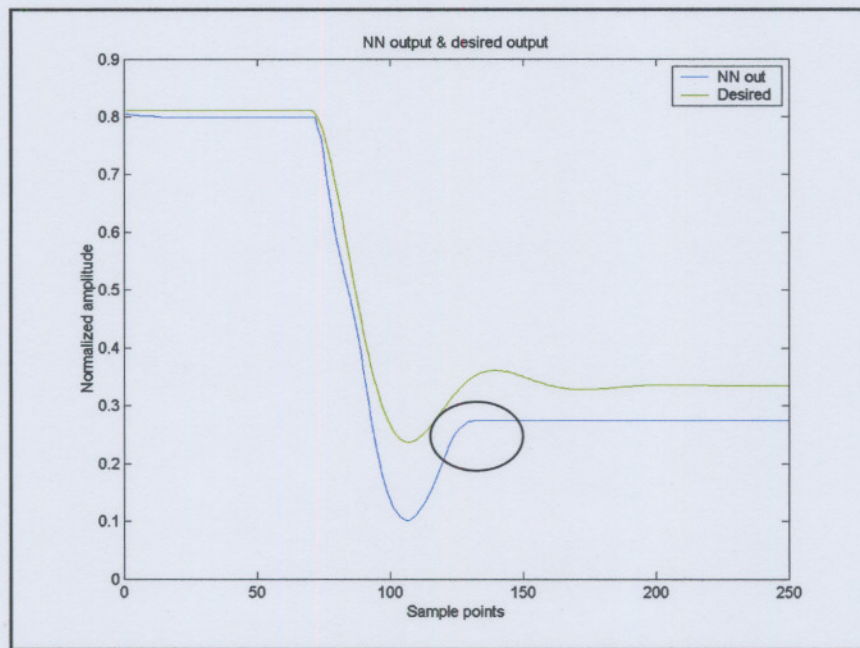


Figure 5.6: Neural network simulation with insufficient delays.

It is clear where the input transfer dynamics left the visual scope of the neural network. This network had 100 delays in the input layer and 60 delays in the second or hidden layer. In this instance, the network did not follow the signal very well. Experiments pointed to the delays in the hidden layer as the *culprit*. The delays in the hidden layer were increased to that of the input layer. The number of delays was determined by the number of sample points from the step of the input to where the ripple settled within a small error of the final value.

INDUCTION MACHINE BENCHMARK

As with the classification architectures before, the more complex neural networks are not necessarily viable solutions. In table 5.3 two network architectures are compared by the mean square error (MSE) of the test samples. The difference lies in the number of neurons in the first two layers. Both networks had 120 time delays in their layers, except for the output layer, which had none.

Architecture	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Total
30-20-1	0.0415	0.1473	0.0330	0.1250	0.1252	0.0346	0.5066
12-10-1	0.0118	0.0318	0.0066	0.0072	0.0263	0.0158	0.0995

Table 5.3: Complex networks are insufficient.

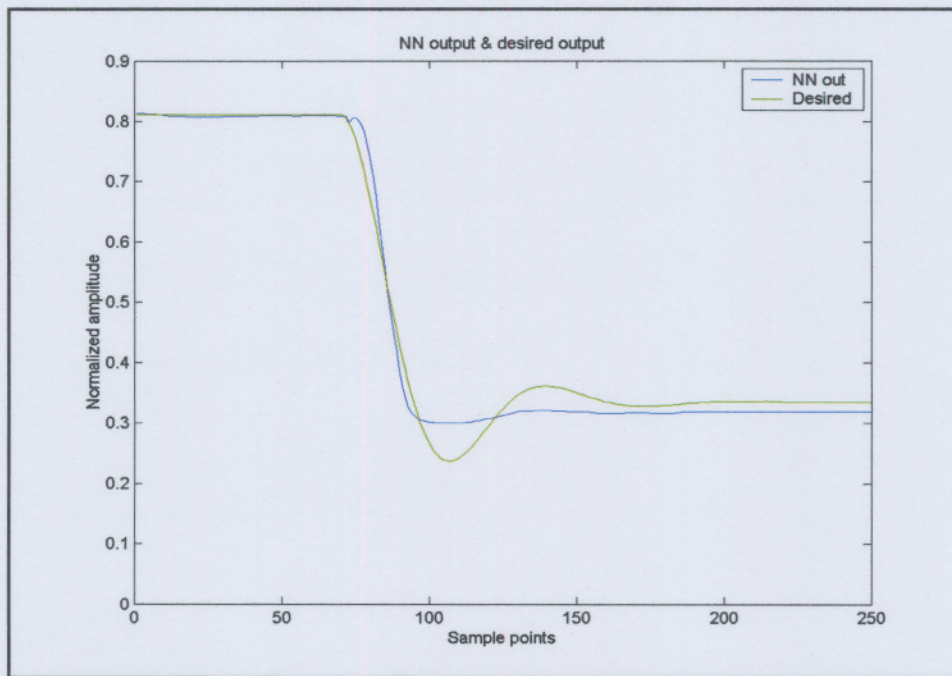


Figure 5.7: The step response of the 30-20-1 network.

For dynamic neural networks, an increase in the number of neurons did not lead to faster training time, as was the case for static neural networks. When one neuron is added, the same number of weights is added as the number of delays in that layer. Figures 5.7 and 5.8 are from sample 3 in table 5.3. In both cases this was the sample that performed the worst out of the whole test set. The numeric and visual data clearly shows that the simpler network outperformed the complex one.

Figure 5.8 illustrates the worst sample of the best performing network and proves that the dynamic neural networks defined by the MATLAB[®] neural network toolbox are sufficient to model non-linear dynamic processes.

INDUCTION MACHINE BENCHMARK

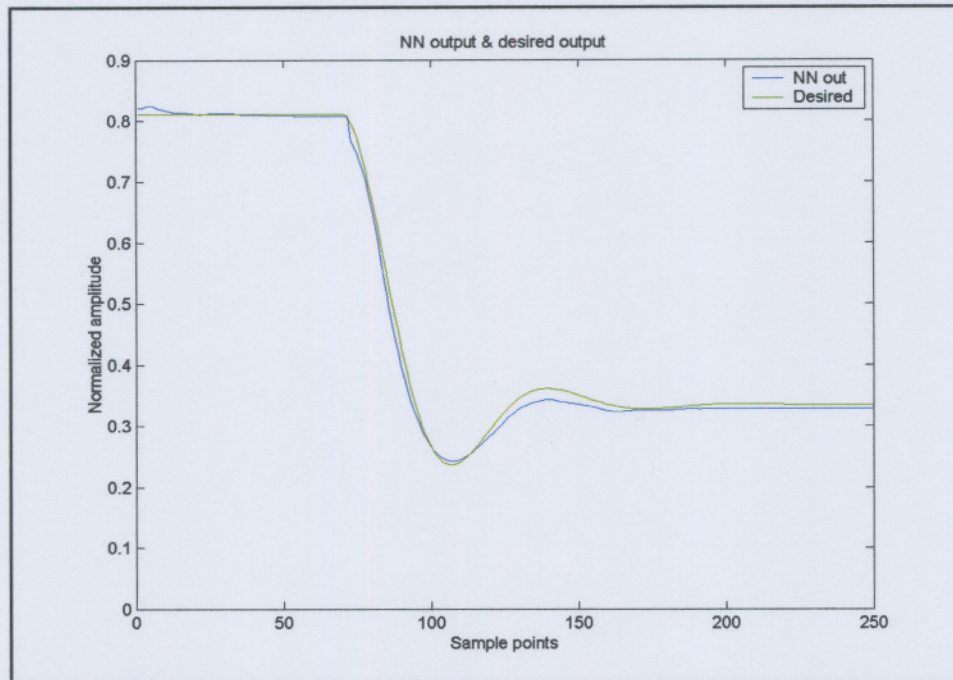


Figure 5.8: The step response of the 12-10-1 network.

5.5 Conclusion

FDD systems that use models of the process to generate residuals need to have models that are as close as possible to the real plant. This is necessary to keep modelling uncertainties from hiding actual plant errors, which is just as undesirable as if the system detects faults that are actually modelling uncertainties.

In this chapter neural networks showed great potential as process-modelling tools. All the equations needed to describe an induction machine become obsolete. Neural networks as modelling tools make development much less time-consuming and cheaper.

6 FLOWNET MODEL OF THE BRAYTON CYCLE

A part of the FLOWNET model of the Brayton cycle is emulated using a dynamic neural network. Instrumentation errors are added and the classification thereof is investigated. The fault detection and detection concept is investigated as a whole in this section.

6.1 Background

6.1.1 Introduction

A schematic representation of the Brayton cycle is given in figure 1.1, section 1.1. This figure illustrates how the Brayton cycle is implemented in the pebble bed modular reactor (PBMR). The pebble bed modular reactor is an initiative by ESKOM, which bought the HTR (high temperature reactor) technology in 1996. ESKOM's motive was to develop this HTR technology into a viable and safe source of electrical power. Advances made in gas turbine technology enabled the use of a direct cycle power conversion system. This is one of the changes made to the technology, making the PBMR unique in comparison with the original German designs (http://www.pbmr.co.za/2_about_the_pbmr/2_8background_to_the_pbmr.htm).

6.1.2 Theory

A programme for simulation and modelling called FLOWNET is used to model the proposed Brayton cycle power plant. Since FLOWNET uses modelling equations that are outside the scope of this paper, simplified phenomenological descriptions of the components are given as background theory. A discussion on FLOWNET version 5.906 is given at the end of this section.

In the Brayton cycle, figure 1.1, there is a high-pressure compressor and a low-pressure compressor. These compressors compress the helium before it enters the core where it will be heated. Helium is used because it is inert to all the materials that are used in the PBMR design. It does not form water droplets in the turbines as with designs using steam and are thus more efficient (https://www.pbmr.co.za/2_about_the_pbmr/2_3how_it_works.htm). The heated gas flows from the core and decompresses through the high-pressure turbine, low-pressure turbine and the power turbine. These turbines are driven by the compressed and heated helium, which in turn drives the high-pressure compressor, low-pressure compressor and the electric generator. The turbines, compressors and heat exchangers are discussed in the following sections. The thermal energy generated by the nuclear reaction is converted to mechanical energy, which in turn is tapped at the generator as electrical energy.

FLOWNET – BRAYTON CYCLE

6.1.2.1 The core

The helium that will be used in the final production plant is heated by the nuclear reaction. This thermal energy increases the gas temperature at constant pressure. The detailed working of the core is beyond the scope of this paper and only the thermal output of the subsystem is considered. The following assumptions are made, which substantially simplifies the model: The outlet temperature is valid throughout the volume of the core and the ambient heat loss is also approximated by this assumption.

The balance equation for this subsystem is given by:

$$(\text{Heat power input}) - (\text{Heat power output}) = (\text{Rate of heat accumulation}).$$

The main heat power input Q_n , would be the heat generated by the nuclear fusion in Watt. The other heat source is the gas flow from the recuperator into the core, $mc_p T_1$. The heat power exits the system via gas flow to the turbines, $mc_p T_2$, and losses to the atmosphere, $\frac{T_2 - T_a}{R_a}$ for this model.

The definitions of these symbols are given in table 6.1. Implementing this with the balance equation gives:

$$Q_n + mc_p T_1 - mc_p T_2 - \frac{(T_2 - T_a)}{R_a} = C \frac{dT_2}{dt}. \quad (6.1)$$

Q_n	Reaction power	Watt
m, m_a, m_b	Mass flow rate	kg/s
c_p, c_a, c_b	Specific heat capacitance of fluid at constant pressure	J/kgK
T_1, T_{a1}, T_{b1}	Inflow temperature	K
T_2, T_{a2}, T_{b2}	Outflow temperature	K
T_a	Ambient temperature	K
R_a	Thermal resistance to ambient	W/K
P_1	Inflow pressure	Pa
P_2	Outflow pressure	Pa
C, C_a, C_b	Thermal capacitance	K/J

Table 6.1: Symbol definitions for the Brayton-cycle model.

FLOWNET – BRAYTON CYCLE

6.1.2.2 The compressors

The following brief note on entropy is necessary to understand some of the concepts used in modelling the compressors and turbines. For a reversible process per unit mass:

$$dQ = du + pdv, \text{ is true,} \quad (6.6)$$

where:

dQ is the energy added in J,

du is the change in internal energy in J,

pdv is the work done by the fluid in J,

p is the pressure in Pa and

dv is the change in volume, m^3 .

For a perfect gas the following equations are applicable:

$$dQ = c_v dT + RT \frac{dv}{v}, \quad (6.7)$$

$$\therefore \frac{dQ}{T} = \frac{c_v dT}{T} + R \frac{dv}{v}, \quad (6.8)$$

$$\therefore \frac{dQ}{T} = \frac{c_v dT}{T} + R \frac{dv}{v} = 0. \quad (6.9)$$

Where:

c_v is the specific heat at constant volume, J/kgK,

dT is the temperature change in K,

T is the temperature in K,

R is the gas constant in J/kgK,

dv is the change in volume, m^3 , and

v is the volume in m^3 .

Equation (6.9) is for an adiabatic process, where $dQ = 0$ and in this case the entropy, s is zero. For

all working substances $ds = \frac{dQ}{T}$ it may also be written as $dQ = Tds$. For a reversible process

$$Q = \int Tds.$$

s is the specific entropy in J/kgK,

FLOWNET – BRAYTON CYCLE

Now, consider a compressor:

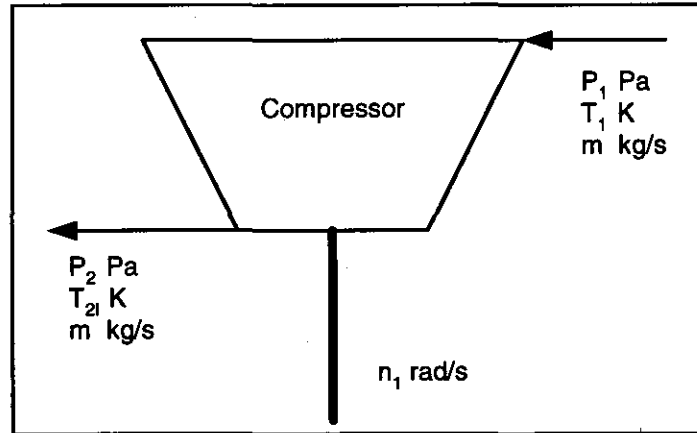


Figure 6.1: Schematic representation of a compressor.

For the compressor model in figure 6.1, phenomenological theory is presented. The position of the inlet and outlet temperatures and pressures are illustrated above. If the process were reversibly adiabatic, the entropy would be constant and the temperature would change from T_1 to T_2 , figure 6.2. If the process were assumed to be polytropic, the temperatures would change from T_1 to T_{2i} .

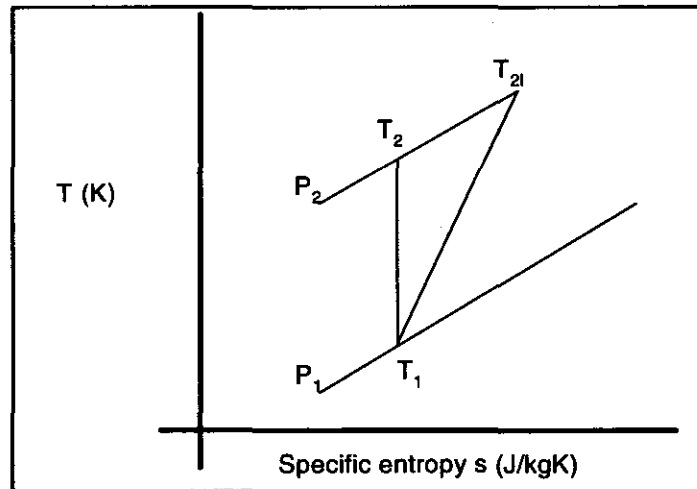


Figure 6.2: T-s diagram for a compressor.

The polytropic temperature equation and polytropic efficiency is respectively given by

$$\frac{T_2}{T_1} = \left(\frac{P_2}{P_1} \right)^{\left(\frac{\gamma-1}{\gamma} \right)} \quad (6.2)$$

and

$$\eta_c = \frac{T_2 - T_1}{T_{2i} - T_1} \quad (6.3)$$

FLOWNET – BRAYTON CYCLE

The power input to the compressor is

$$P_c = mc_p (T_{2i} - T_1), \quad (6.4)$$

with c_p the specific heat at constant pressure and T_{2i} is given by,

$$T_{2i} = T_1 + \frac{(T_2 - T_1)}{\eta_c}. \quad (6.5)$$

A more complete model would use characteristic curves of the compressor to solve for the mass flow rate, temperature and pressure. The system would have flow resistance and fluid capacitance and will be a multiport model.

6.1.2.3 The turbines

This turbine model is also based on phenomenological theory, where a more complete multiport model would use characteristic curves. The same assumption that this is a polytropic process is made and thus the temperatures in figure 6.3 will go from T_1 to T_{2i} illustrated by figure 6.4.

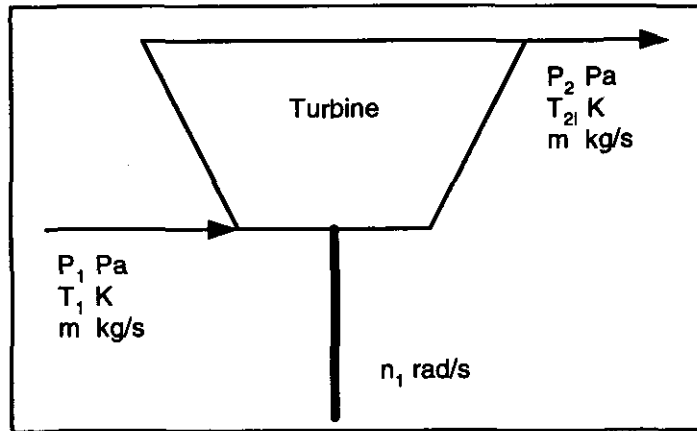


Figure 6.3: Schematic representation of a turbine.

For the turbine, the temperatures and pressures are related by the following polytropic equation:

$$\frac{T_2}{T_1} = \left(\frac{P_2}{P_1} \right)^{\left(\frac{\gamma-1}{\gamma} \right)}. \quad (6.10)$$

The polytropic efficiency is given by

$$\eta_t = \frac{T_1 - T_{2i}}{T_1 - T_2}, \quad (6.11)$$

where

$$T_{2i} = T_1 - \eta_t (T_1 - T_2). \quad (6.12)$$

FLOWNET – BRAYTON CYCLE

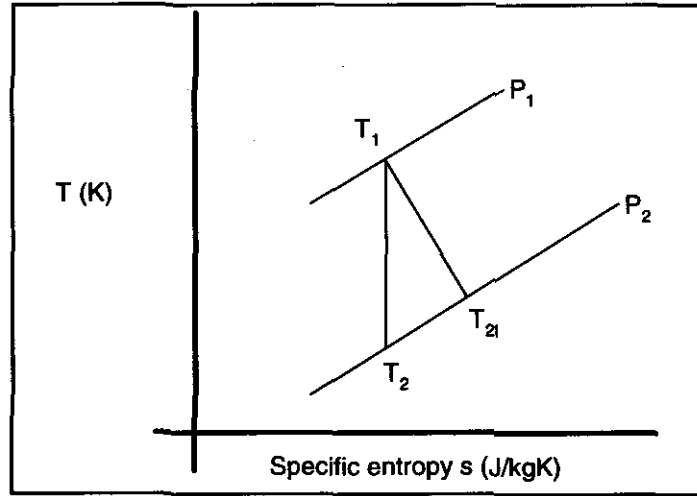


Figure 6.4: T-s diagram for a turbine.

The power output can then be calculated by

$$P_t = mc_p (T_1 - T_{2i}), \quad (6.13)$$

where c_p is the specific heat at constant pressure.

6.1.2.3 The heat exchanger.

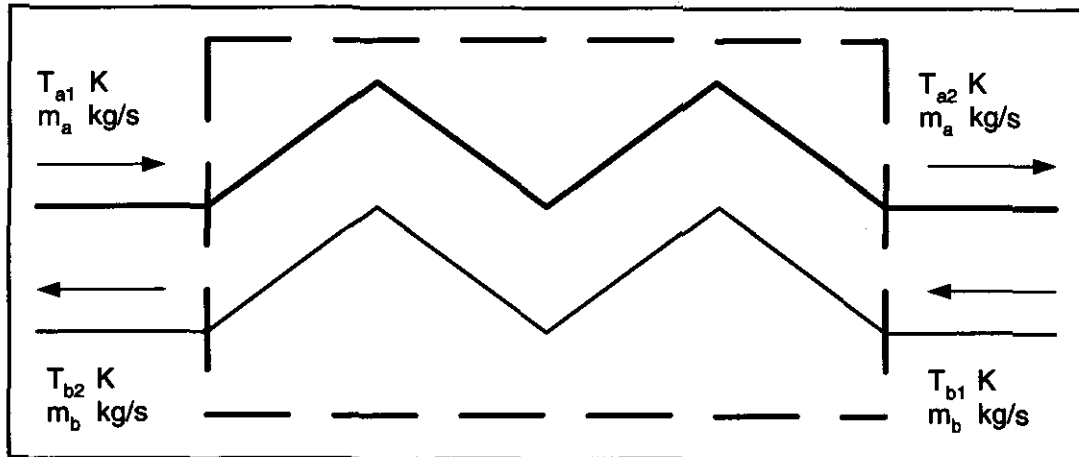


Figure 6.5: Schematic representation of a heat exchanger.

Figure 6.5 is a schematic representation of a simplified concurrent heat exchanger model. More accurate heat exchanger models would use the logarithmic mean temperature difference. The heat balance equation for side a , which is the canal that flows from left to right in figure 6.5, is given by:

$$(\text{Rate of heat inflow}) - (\text{rate of heat outflow}) = (\text{rate of accumulation of heat})$$

or

$$\frac{1}{R} (T_{b2} - T_{a2}) + m_a c_a T_{a1} - m_a c_a T_{a2} = C_a \frac{dT_{a2}}{dt}. \quad (6.14)$$

FLOWNET – BRAYTON CYCLE

For side b , flowing from right to left, this equation is:

$$-\frac{1}{R}(T_{b2} - T_{a2}) + m_b c_b T_{b1} - m_b c_b T_{b2} = C_b \frac{dT_{b2}}{dt}. \quad (6.15)$$

The definitions of the variables in these equations are given in table 6.1.

6.1.2.5 Flownet

Flownet is a general network analysis code (<http://www.flownet.net/frame/default.html>) that solves the flow, pressure and temperature distribution in large arbitrary-structured thermal-fluid networks. This software has a library of predefined blocks that represents components such as pipes, compressors, turbines, heat exchangers, valves and components needed to simulate many industry based thermal-fluid processes. It can deal with both gases and fluids and is able to simulate steady state and transient flows.

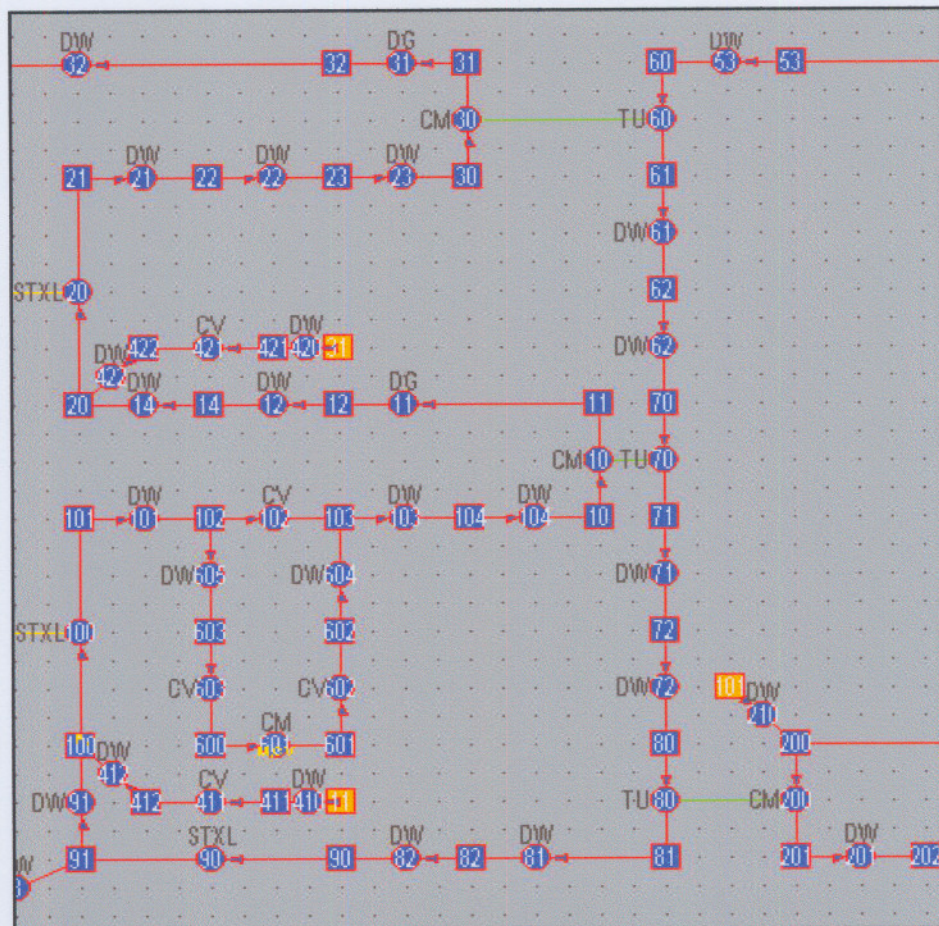


Figure 6.6: Flownet model schematic.

FLOWNET – BRAYTON CYCLE

It uses a graphical user interface and a network can be drawn by connecting existing blocks representing the different components. Figure 6.6 is an example of a network. This section includes the three turbines in the Brayton cycle, elements (circles) 60, 70 and 80. The two compressors are represented by elements 30 and 10.

6.2 Methodology

6.2.1 Modelling

6.2.1.1 Brayton cycle

In this section the modelling and classification concepts investigated above are applied to part of the Brayton cycle. First the modelling part is investigated, and then the classification part. For the modelling, Flownet is used to generate simulation data of nitrogen insertion and extraction in the PBMM Brayton cycle. The pressure is taken at node 60, 61, 71 and 81 in figure 6.6, which represents input and output pressures for each of the three turbines. A node is represented by a square. The high-pressure turbine is between node 60 and 61, the low-pressure turbine is between node 61 and 71 and the power turbine is between node 71 and 81. Note that the effects of the pipes between the turbines are not considered in this paper.

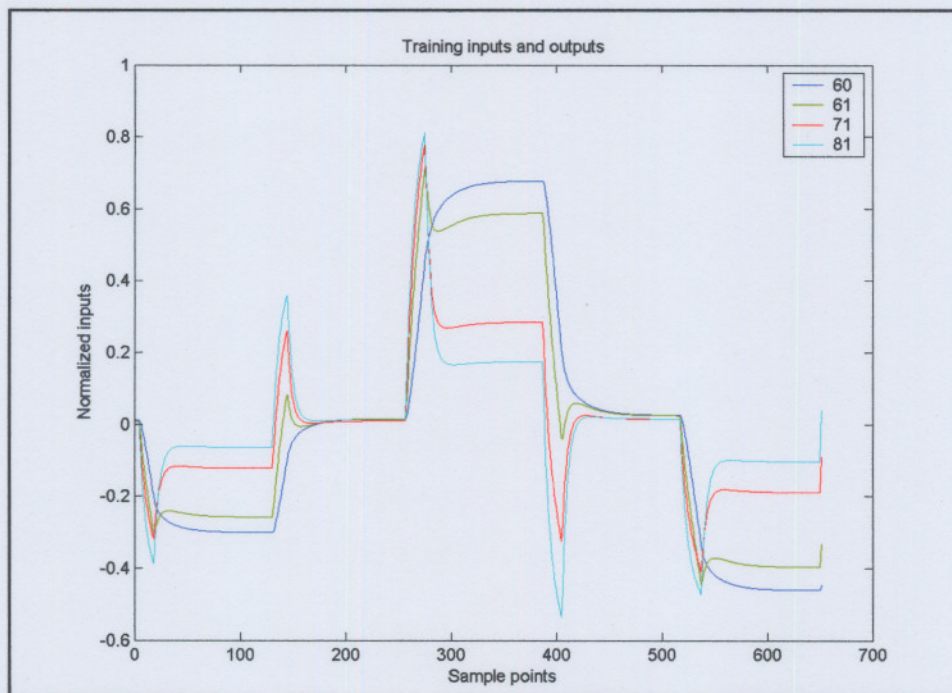


Figure 6.7: Input/output samples.

FLOWNET – BRAYTON CYCLE

Flownet has a SIMULINK[®] interface that is used to generate the insertion and extraction signal for input to the Flownet model. This interface is also used to store the simulated data in the MATLAB[®] workspace from where the samples can be grouped and manipulated. The training data is presented to the network in the form of a single long time series. The time series consists of 150 different gas inlet and outlet simulations. Figure 6.7 illustrate 4 signals that have been generated from the Flownet simulation.

6.2.1.2 FDD system

In practice, an FDD system will simulate all the processes or subsystems either individually or grouped with related or *neighbouring* subsystems. Parts of a process can be taken as subsystems if measurements could be taken before and after that section. Although detectability and isolability as discussed in chapter 4 are increased when more subsystems are modelled, there is a trade-off between these factors and computation time. If a process could be divided into numerous subsystems, the optimum solution might not be by using each possible section.

An efficient, robust, low-maintenance and low-cost FDD system is the ideal. In this paper only three *neighbouring* turbine systems were investigated to develop an FDD system concept. Detectability and isolability are also dependent on how residuals or groups of residuals are manipulated and integrated. A concept where three related subsystems are modelled and integrated is investigated in this section.

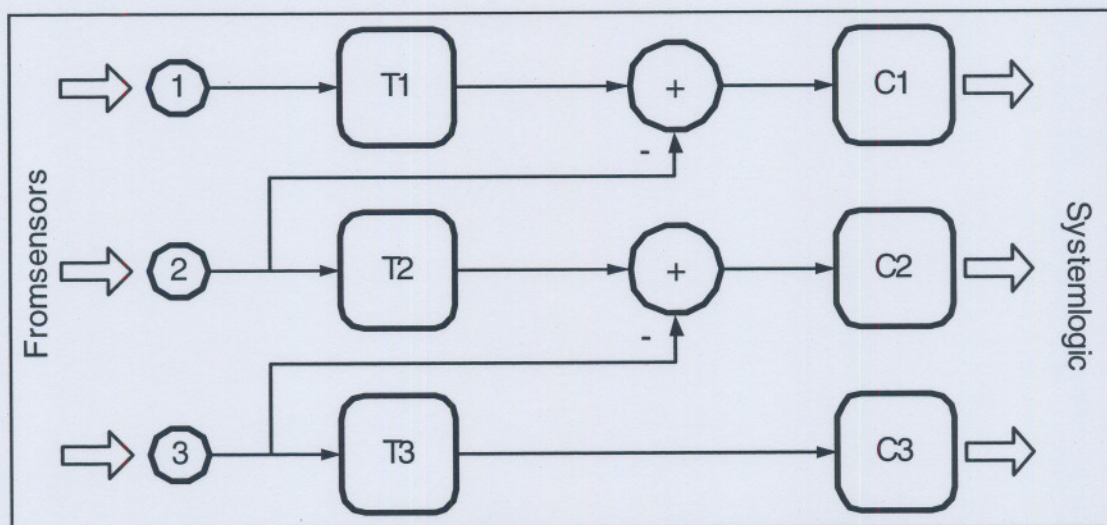


Figure 6.8: FDD concept on part of the Brayton cycle.

FLOWNET – BRAYTON CYCLE

Figure 6.8 illustrates a part of what conceptually would make up an array of neural networks. This array of neural networks models some components of the Brayton cycle. T1, T2 and T3 are the neural network models of the three turbines, while C1, C2 and C3 are the classification networks that classify the residuals to certain fault classes. The inputs to the neural networks come from the sensors on the real plant at their respective positions. This layout of networks was chosen from three candidates. Only the three subsystems modelled in the previous section were considered. These layouts are shown in figure 6.9, where the gas flow is from left to right. There are three subsystems: T1, T2 and T3. Each horizontal square bracket represents a neural network with its input and output taken at the left- and right-hand point of that bracket.

Layout A has three networks with a common input at the left-hand side. However each successive network has an extra subsystem to model and the complexity increases as more subsystems are included. For three subsystems the 3rd network must simulate all three subsystems with their combined non-linearities. For the whole Brayton cycle this layout would lead to networks having to simulate highly complex combinations of sub systems.

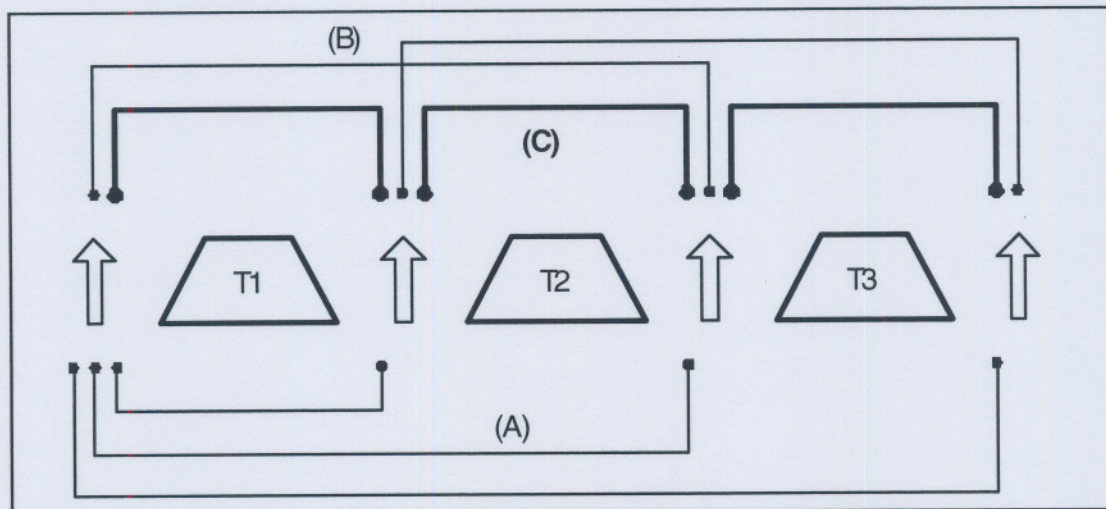


Figure 6.9: Three neural network layout combinations.

Layout B has two networks for three sub systems, while the networks must each simulate two subsystems. This leads to more complex networks, while one still only saves one network for a process with many more sub-systems. Layout C used in this investigation has a network for each subsystem. The reason behind choosing C is that less complex processes can be modelled more accurately and the training time would be less. This layout will not be sufficient when the output of one network serves as the input for the following network. This is because errors propagate to the next network and could explode.

FLOWNET – BRAYTON CYCLE

6.2.1.2 Modelling results

More training methods were investigated and it was found that the *trainbfg* algorithm performed best compared to the other two algorithms below in terms of the number of epochs and accuracy.

		100 epochs	500 epochs	1500 epochs
<i>trainrp</i>	- 1	0.00163	0.00187	0.00071
	- 2	0.000655	0.000241	0.00023
	- 3	0.000378	0.000249	0.00019
<i>traingdx</i>	- 1	0.0055	0.00085	0.00058
	- 2	0.0035	0.00038	0.00029
	- 3	0.0557	0.00042	0.00027
<i>trainbfg</i>	- 1	0.000367	0.00019	0.000063
	- 2	0.000147	0.00011	0.000107
	- 3	0.000214	0.00016	0.000143

Table 6.2: Accuracy (MSE) and speed of the three training algorithms investigated.

The *trainbfg* algorithm trained the fastest and the most accurate in terms of mean-squared-error. The following figure illustrates the first neural network modelling the first turbine's pressure output according to Flownet. The input is the blue signal in figure 6.7.

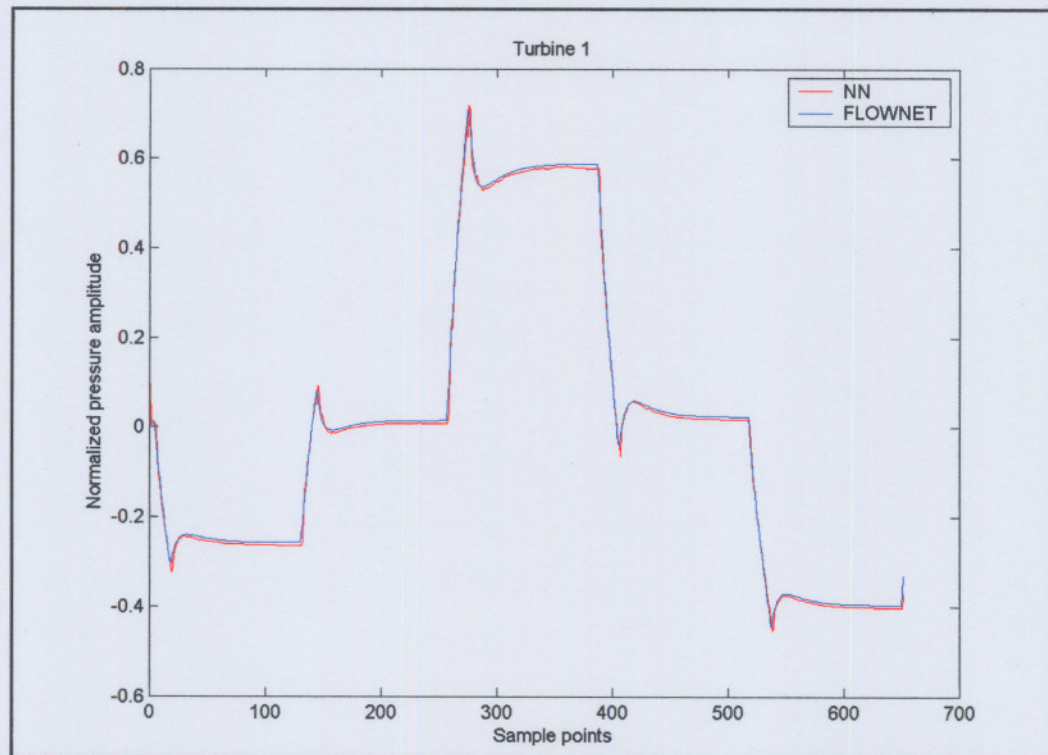


Figure 6.10: The modelling capabilities of these neural networks.

FLOWNET – BRAYTON CYCLE

The architecture of all three of these networks was quickly found. The method was to start with a simple network, increasing the complexity until the optimum architecture was obtained. All three networks have a 2-1 topology, two input neurons and one output neuron, and differ only in the number of input weights. The networks for turbine 1 and 2 both have 90 time delays and the network for turbine 3 has 100. The number of weights was established by starting with 5 weights and increasing that number until an optimum MSE was produced.

The problem that occurred when the number of weights was insufficient is illustrated in figure 6.11. The offset error inside the circle decreases to the right as the number of weights increases. The overshoot on the left-hand side of the circle disappears as well. The overshoot might be explained by viewing the rectangle as the *visual scope* of the neural network.

When the network trains the overshoot in the training data, *blue*, it faces overshoots going both ways. When coming down from the first overshoot, the steep descent or ascent might cause the network to simulate an overshoot of the opposite direction. As the number of weights increases, the network can *see* more of the signal and will *know* it has already simulated the one overshoot.

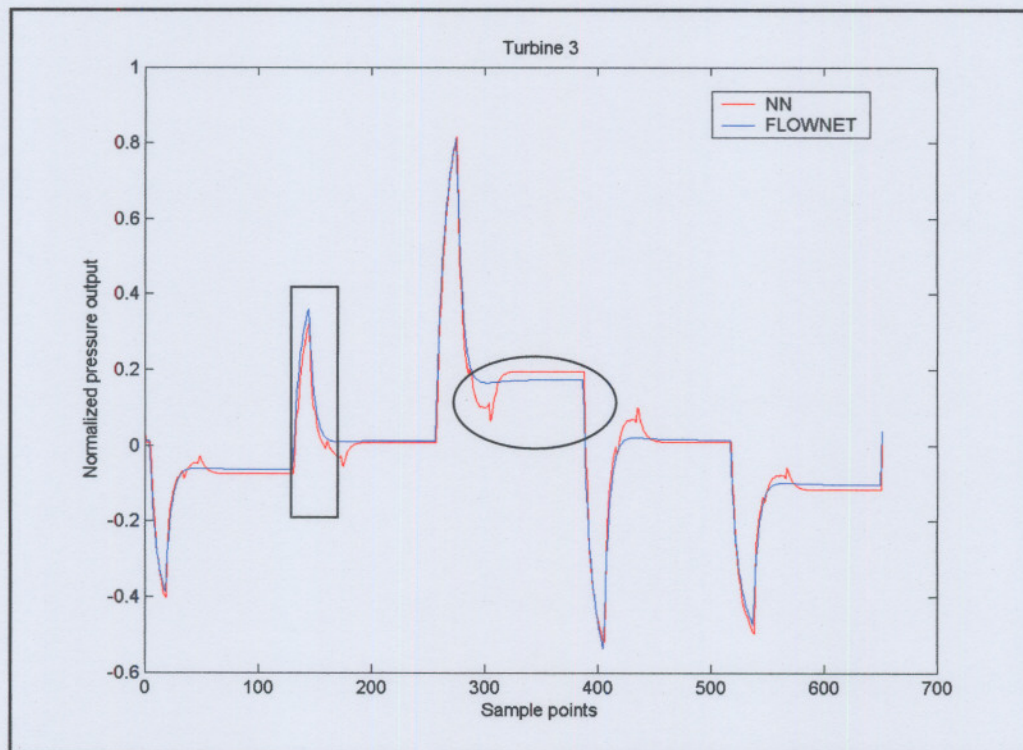


Figure 6.11: Illustration of insufficient weights error for this scenario.

FLOWNET – BRAYTON CYCLE

Another observation that can be made when the number of weights neared the optimum is that both the overshoot and most of the offset error disappeared. The network still generates some ripple, which only ceases when a sufficient number of weights have been defined. It can be concluded that the number of weights does have an effect on the ability of the network to *fit* or *map* the output correctly and smoothly.

6.2.2 Classification

6.2.2.1 Concept

In this part of the FDD concept the assumption is made that the first sensor cannot be faulty. Referring to figure 6.8, this means only sensors 2 and 3 may have an error in this instance. A faulty first sensor will be considered in a later section.

A sensor 2 error causes an error in the first residual and not only in its own second residual. Similarly an error in the third sensor works its way into the second residual as well. The end result is that when sensor 2 becomes faulty both classification networks C1 and C2 will produce outputs that imply sensor errors. The final diagnosis can now easily be obtained by implementing basic logic programming. For example, when C2 and C3 generate an error, the logic will determine that the third sensor is faulty.

To reproduce process errors with the Flownet software package was regarded as too complex and was left outside the scope of this investigation. The principle remains the same, however, because the error may be attributed in such a case to either the sensor or the associated process.

6.2.2.2 Classification of data

In this section classification is done by using dynamic neural networks. A dynamic classifying network can be integrated without creating fault windows, as shown in chapter 4. In a real-time FDD package this would improve efficiency. The residuals were generated by adding offset values of different amplitudes to existing simulated data. In this instance only data *originating* from sensors 2 and 3 are given errors, following the assumption in the previous section. For each of these sensor faults two residuals are generated.

FLOWNET – BRAYTON CYCLE

As an example, the first of the two residuals is caused by the difference of sensor 2 and the output of T1 (See figure 6.8). In the second residual, the difference is caused by the faulty output of T2 because its input has an error in this case. Figure 6.12 is an example of two of these residuals. The spikes that are indicated by the circle come from the deviations between the modelling neural network and the Flownet model. Although they are small relative to the residual caused by T2 in this case, they did pose some classification problems. These problems are discussed in the next section.

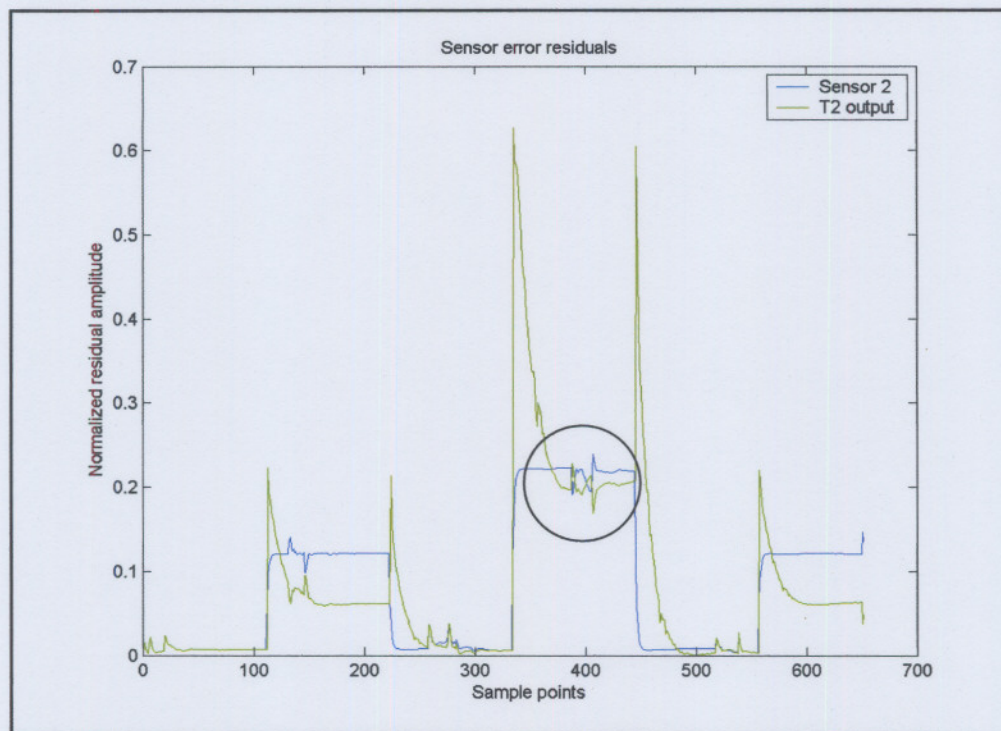


Figure 6.12: An example of sensor fault residuals.

6.2.2.3 Classification results

The dynamic classification neural network is presented with the training data in one long signal that over time consists of the different possible inputs. The second and third classification networks have two sets, each being a long time series of possible inputs and outputs. The sensor errors, the blue in figure 6.12, are much the same for all three classification networks, while the faulty outputs for T2 and T3 in figure 6.8 are also similar. This made the finding of a suitable topology easier and faster, which will be shown below.

The major problem was the modelling deficiencies that caused the faulty classification. These faulty classifications lasted only for short instances, as illustrated in figure 6.13. A filter was implemented to remove most of the spikes from these modelling deficiencies.

FLOWNET – BRAYTON CYCLE

What was left is circled in figure 6.12. This left the network to have to act as a filter while classifying the residuals. For the first classification network C1 in figure 6.8, short misclassifications could not be entirely removed. The solution, however, lied with C2 and C3 where their additional input training set consisted of much more spiky data. This forced the classification network to also act as a kind of filter.

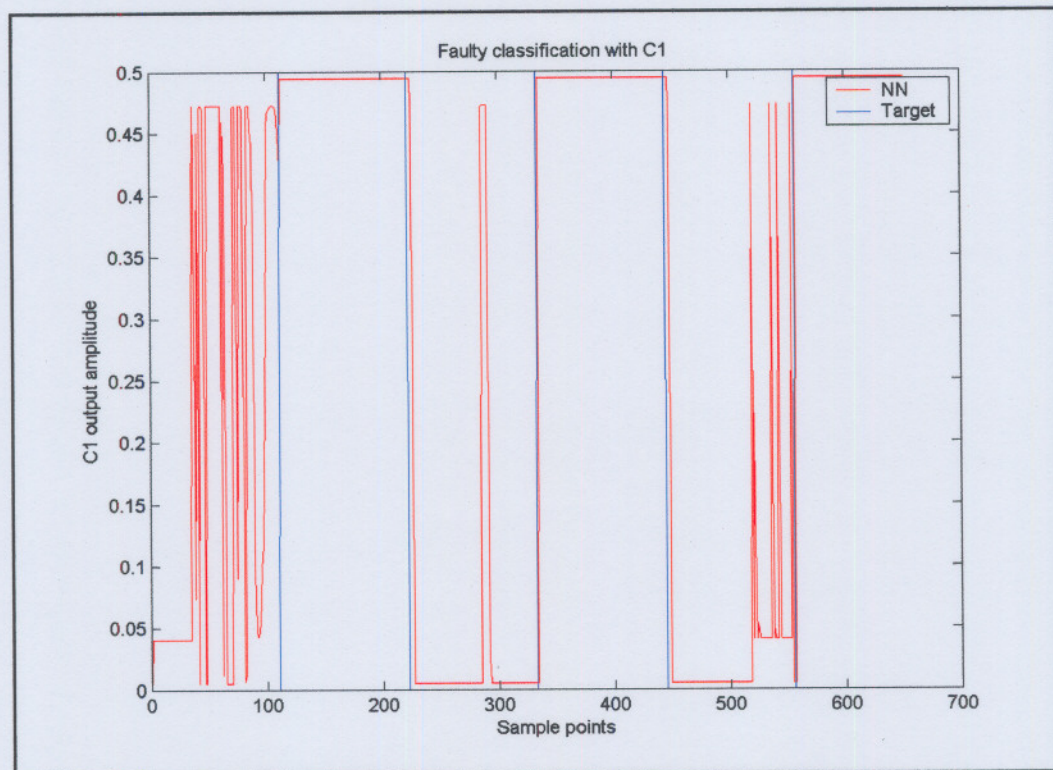


Figure 6.13: Misclassifications of the first classification network, C1.

The final network of C2 was tested with the data for C1 and C3 and passed with a low number of errors. The same network that was used for the second classification neural network in this FDD concept was also used for the classification problems of C1 and C3.

The method of “start simple and increase complexity if not sufficient”, was used again to obtain the best architecture. The two-layer input and output layer topology did not perform well. The noise proceeded through the network without being *filtered* or manipulated. The difference can be seen by comparing figure 6.13, a 3-layer network, with figure 6.14 which illustrates the output of a 10-1 network topology. The 3-layer topology tried to classify the noise by generating short pulses in figure 6.13, while the 2-layer network’s attempt is visibly inferior in figure 6.14.

FLOWNET – BRAYTON CYCLE

Although the network of figure 6.14 was trained with data that has more spikes, the 3-layer network still gives better results, even though it was only trained with the smoothed sensor residuals. Both these outputs give incorrect classifications, but it does illustrate that this problem would be better solved by the 3-layer topology.

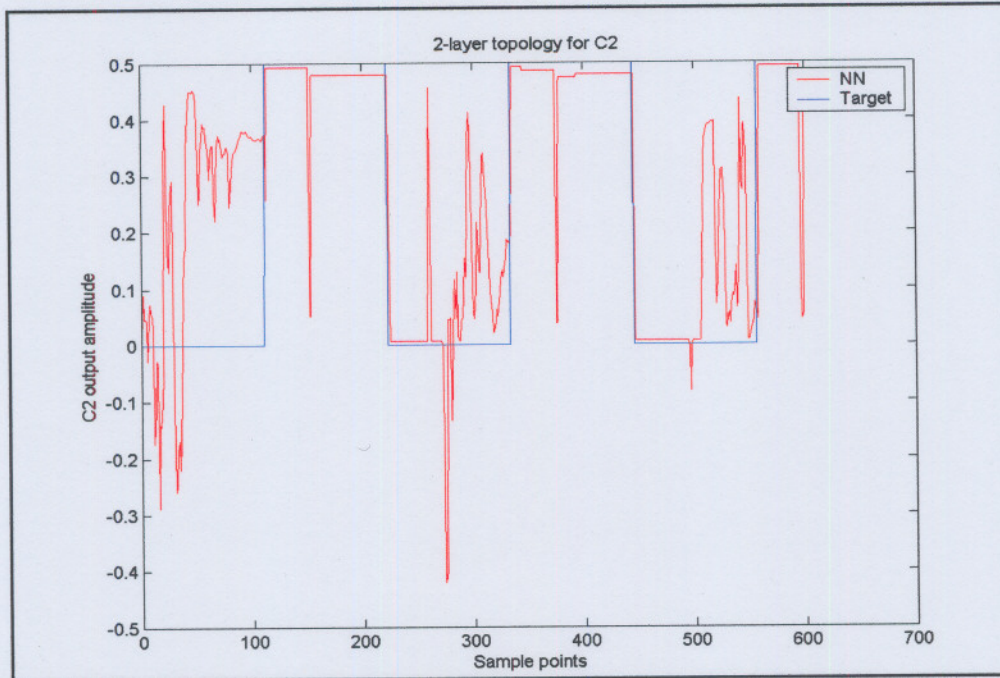


Figure 6.14: Output of a 2-layer topology using both training sets.

To illustrate the influence that the second training set has on the performance of the classification network, the performance of the network with different training algorithms and epochs is tabulated below. The MSE of the networks is first given on the left-hand side if they are trained with the normal sensor set first (set 1) and on the right-hand side the networks are trained with the faulty output set (set 2) first.

	Set 1	Set 2		Set 2	Set 1
50 epochs			50 epochs		
<i>traingdx</i>	0.0361	0.0514	<i>traingdx</i>	0.0724	0.0681
<i>trainrp</i>	0.087	0.021	<i>trainrp</i>	0.0215	0.00726
<i>trainbfg</i>	0.0109	0.022	<i>trainbfg</i>	0.0251	0.00865
100 epochs			100 epochs		
<i>traingdx</i>	0.024	0.0411	<i>traingdx</i>	0.54	0.020
<i>trainrp</i>	0.0081	0.0191	<i>trainrp</i>	0.01596	0.00634
<i>trainbfg</i>	0.0082	0.0194	<i>trainbfg</i>	0.0227	0.00827
500 epochs			500 epochs		
<i>traingdx</i>	0.0104	0.0282	<i>traingdx</i>	0.0293	0.0101
<i>trainrp</i>	0.00783	0.0109	<i>trainrp</i>	0.01349	0.00583
<i>trainbfg</i>	0.00607	0.0192	<i>trainbfg</i>	0.00879	0.00578

Table 6.3: Effect of the extra training set on classification performance.

FLOWNET – BRAYTON CYCLE

The bold-printed values are the smallest MSE in all the groups of three different training algorithms. The highlighted values are the smallest MSE when comparing each set on the left-hand side of table 6.3 with its corresponding set on the right.

It is clear that these two training sets compliment each other. However, what is the real advantage? The classification output in figure 6.13 is the network used for C1 in figure 6.8, which is a 4-4-1 topology that was trained with only the sensory residuals. This network had an MSE of 0.00449, but when compared with figure 6.15, it has short misclassifications right through the test set. The network topology of figure 6.15 is 6-4-1 and was trained by both training sets. This specific network has an MSE value of 0.00578. This error is larger than that of the network of figure 6.13, but the classification is visibly better and more stable. The major factor that made the MSE value larger is the ± 0.04 offset errors shown in figure 6.15, which is minimal in figure 6.13. This offset error can easily be compensated for while the misclassifications in figure 6.13 will not be that easy, resulting in some falsely positive indications.

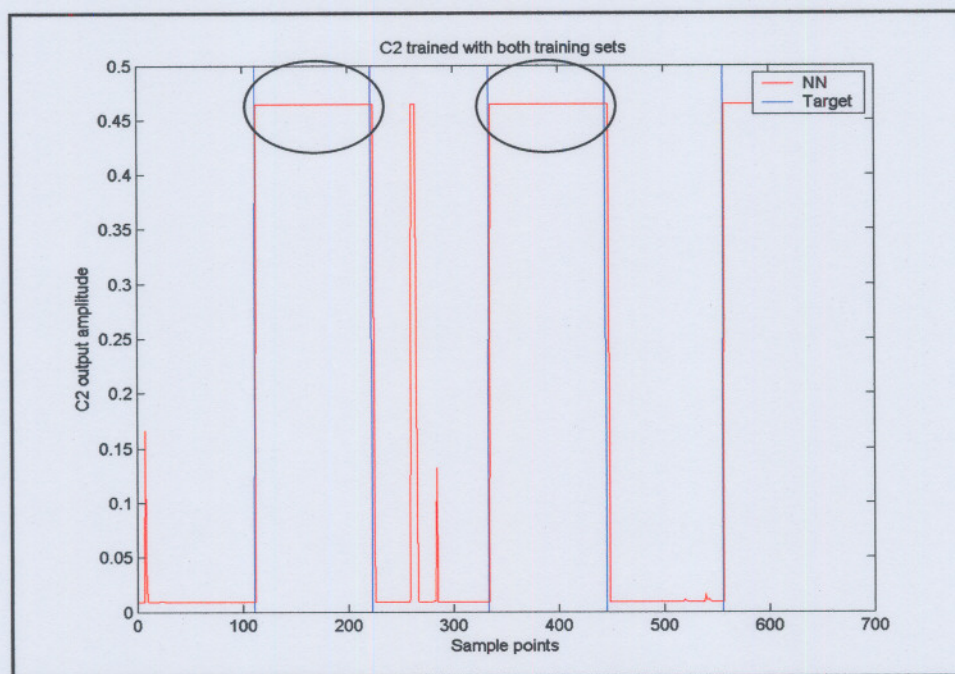


Figure 6.15: Classification network trained with both training sets.

The real advantage of the extra noisy set is visible in this example in that it makes the neural network much less susceptible to noisy data. By implementing this second noisy set in the training of the C1 neural network in figure 6.13, the classification performance was comparable to the C2 neural network of figure 6.15.

FLOWNET – BRAYTON CYCLE

Finally, the classification network C2 was tested with the C1 and C3 data and the performance was good right through. This also illustrates the generalising ability of neural networks.

6.2.3 Decision logic

6.2.3.1 Methodology

The function of decision logic in this paper is to take the output of the classification neural networks of the previous section, reduce misclassification and generate a decision on the present fault. To reduce misclassifications, the short incorrect classification pulses in the classification outputs of the previous section were eliminated. Every x data points of the signals were monitored for errors, which brings back the *window* concept. The *window* width is by default 30 data points in this instance, but could be edited in the final programme. Singular misclassification pulses are not problematic unless the chosen *window* width is too small. Problems arise when these misclassifications are bunched together.

Before these problems are discussed, a description is given of how the outputs of the classification networks are changed into a form on which decision logic can be applied. The average over the *window* is taken and must be over $y\%$ of the positive classification value to produce an error. The default value of y in this instance is 80%. This is not the only criterion, however, in each *window* the number of times the classification output changes is monitored. When this number exceeds the value z , no sensor error is reported. These misclassifications occurred readily because in this final integration normally distributed noise was added to the sensor data. The noise has a maximum of 0.2% of the total amplitude of a gas inlet-outlet pair.

Although the neural networks simulating the turbines were also given noisy data to train, their performance was influenced negatively. This may be improved through better training with noisy data. The residuals contained much more noisy data and this caused the classification networks to generate more misclassification pulses. However, these pulses can easily be identified as being caused by noise and the final classification results were acceptable. They may also be reduced due to the threshold that is implemented when the residuals are generated, which removes all the noisy data below this threshold value.

FLOWNET – BRAYTON CYCLE

If the *window* width were changed, the “classification output changes” limit and the average might affect the end result differently. If the size were increased, the number of classification changes would be more. If the fault begins 20% into the *window*, the fault might be detected later than its actual occurrence because the average will not exceed 80% for instance. This could lead to some misclassifications. When one makes the *window* too small, the misclassification pulses lose their pulse character and become more determined. One or two misclassification pulses can now exceed the average and the logic will give a positive incorrect classification. As will be seen in the results for this data, the window width of 30 data points worked well in this investigation.

6.2.3.2 Applied logic

Decision logic for a single fault is first investigated. With three sensors, multiple fault detection is more complex and is discussed later on. The logic is simple: If residual 1 alone indicates a fault, sensor 1 is faulty. If both residuals 1 and 2 indicate faults, sensor 2 is faulty with the assumption that faults do not occur simultaneously. Sensor 3 is faulty when residuals 2 and 3 indicate faults. The *truth-table* is given below.

	Sensor 1 error	Sensor 2 error	Sensor 3 error
Residual 1	1	1	0
Residual 2	0	1	1
Residual 3	0	0	1

Table 6.4: Single sensor error truth table.

For a scenario where there are multiple sensor errors, the residuals that have already been affected will not give extra information when another error occurs. There are some variations that have not been investigated in this paper. Training the network to detect them could be investigated in a future research study.

Some multiple errors may be detected with normal logic. When a more complete system of sensors is taken, the efficiency would be better than with the 3-sensor scenario investigated here. When a single error has occurred, the logic keeps on monitoring the other residuals. The *truth-table* below illustrates the logical problem faced with multiple errors.

	Sensor 1&2	Sensor 1&3	Sensor 2&3
Sensor 1	S2	S3	X
Sensor 2	X	X	S3
Sensor 3	X	S1/S2	S1/S2

Table 6.5: Multiple sensor error logic table.

FLOWNET – BRAYTON CYCLE

Because the probability that the third sensor will fail before the other two have been repaired is small. The logical assumption is made that three sensors cannot break down simultaneously in this investigation. There is still the problem when sensor 1 breaks after sensor 2 because the logic will not be able to identify it. With the problem defined as in figure 6.8 the top sensor will always cause an uncertain double error.

Another problem occurs when sensors 2 and 3 break down simultaneously. When sensor 3 fails first the logic will classify a sensor 2 failure as a sensor 1 failure. In this instance the accuracy will only be tested when sensor 2 breaks before sensor 3.

6.2.3.3 Integrated concept

A MATLAB® GUI or graphical user interface was developed to investigate the efficiency of integrating all the above concepts. The GUI allows the user to select a sensor error or combination of sensor errors and view a scenario of that combination. Figure 6.16 is a screenshot of the GUI.

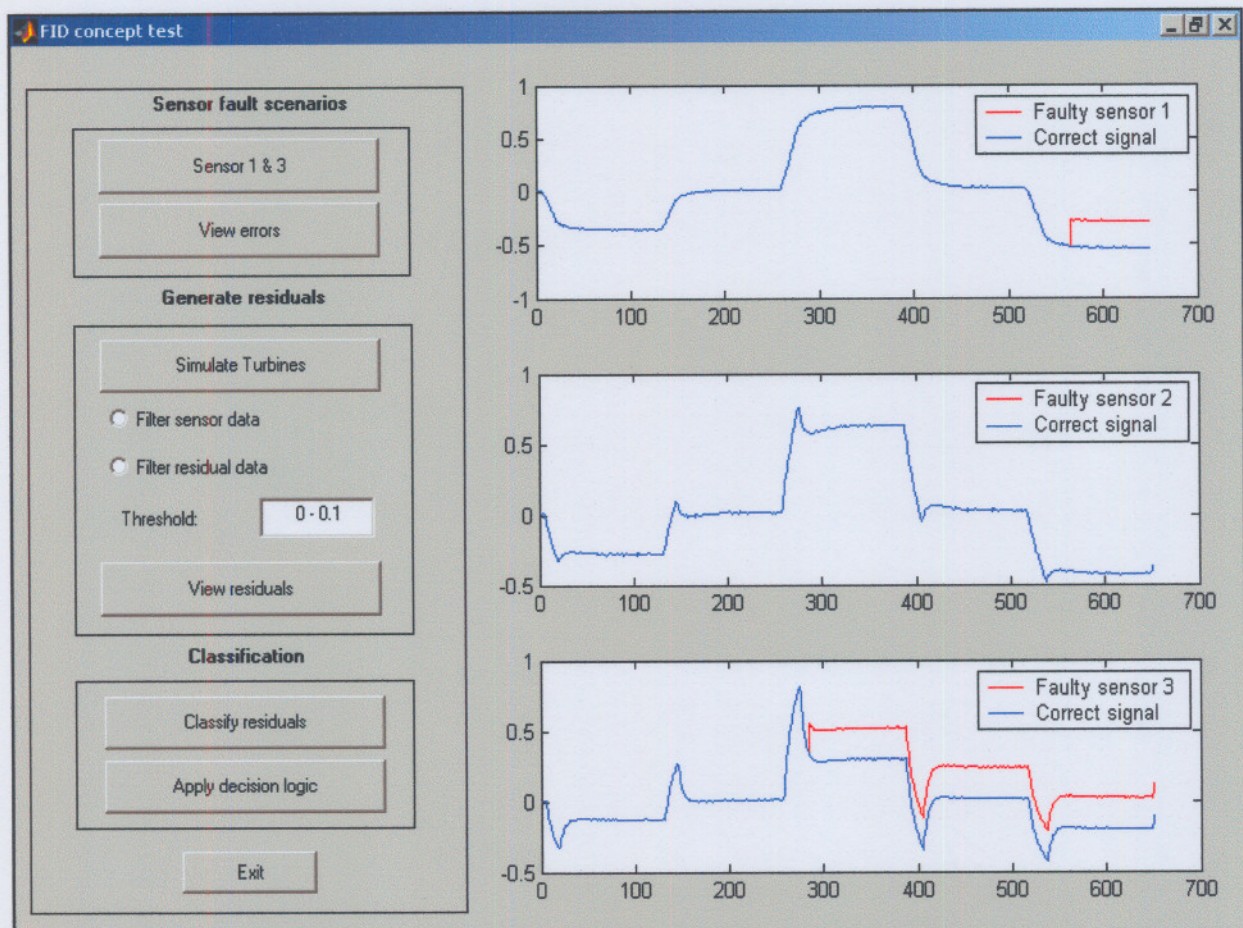


Figure 6.16: MATLAB® GUI developed to integrate all the FDD concepts.

FLOWNET – BRAYTON CYCLE

The GUI also allows one to set the threshold value and to control whether the sensor data and/or the residual data must be filtered. The turbines can then be simulated and this function also generates the residuals. These residuals can be viewed. The classification networks will classify the residuals, while their outputs are automatically displayed. When the decision logic is applied another window is opened and there the *window* width can be set before the final result is given (See figure 6.17).

6.2.3.4 Decision logic results

Firstly, the single errors were investigated. Every sensor fault scenario was run 33 times and the results are as follows: Sensor 1-faults were identified and correctly classified 100%; sensor 2-faults 91% while sensor-3 errors were 97% successfully detected and classified. Sensor-2 fault residuals were more susceptible to noise. This is also evident in the two-sensor faults.

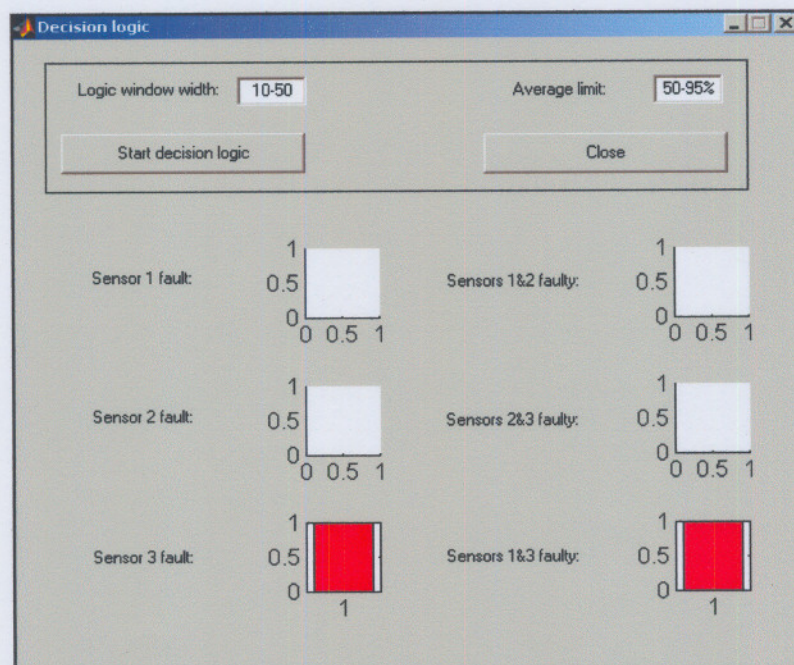


Figure 6.17: An example of a sensor 1 and 3 fault with sensor 3 detected first.

The results of the two-sensor faults were as follows: sensor 1 and 3 were 94% correctly identified, sensor 1- and 2-faults and sensor 2- and 3-faults were both 91% correctly identified. Sensor 2- and 3- faults gave problems and were only 70% successfully classified before the average limit was decreased to 60% from 80% default. As with the single errors, the errors involving sensor 2 residuals gave the most problems in terms of being correctly classified. An example of the classification part of the MATLAB® GUI is given below. Further remarks are given in the final conclusion later in this section.

6.3 Measured data

To test the PBMR concepts, a small scale model was built. The heat source in this instance is an electric heater instead of the nuclear reaction that would ultimately provide all the thermal energy. This project was aptly named the pebble bed micro model or PBMM. Measured data was taken from the PBMM tests.

Data was obtained from a start-up to shut-down test run. This limited the data available for training to sections of the active process. Nevertheless, a test was done on data generated during a load-rejection test. This data is shown below in figure 6.18.

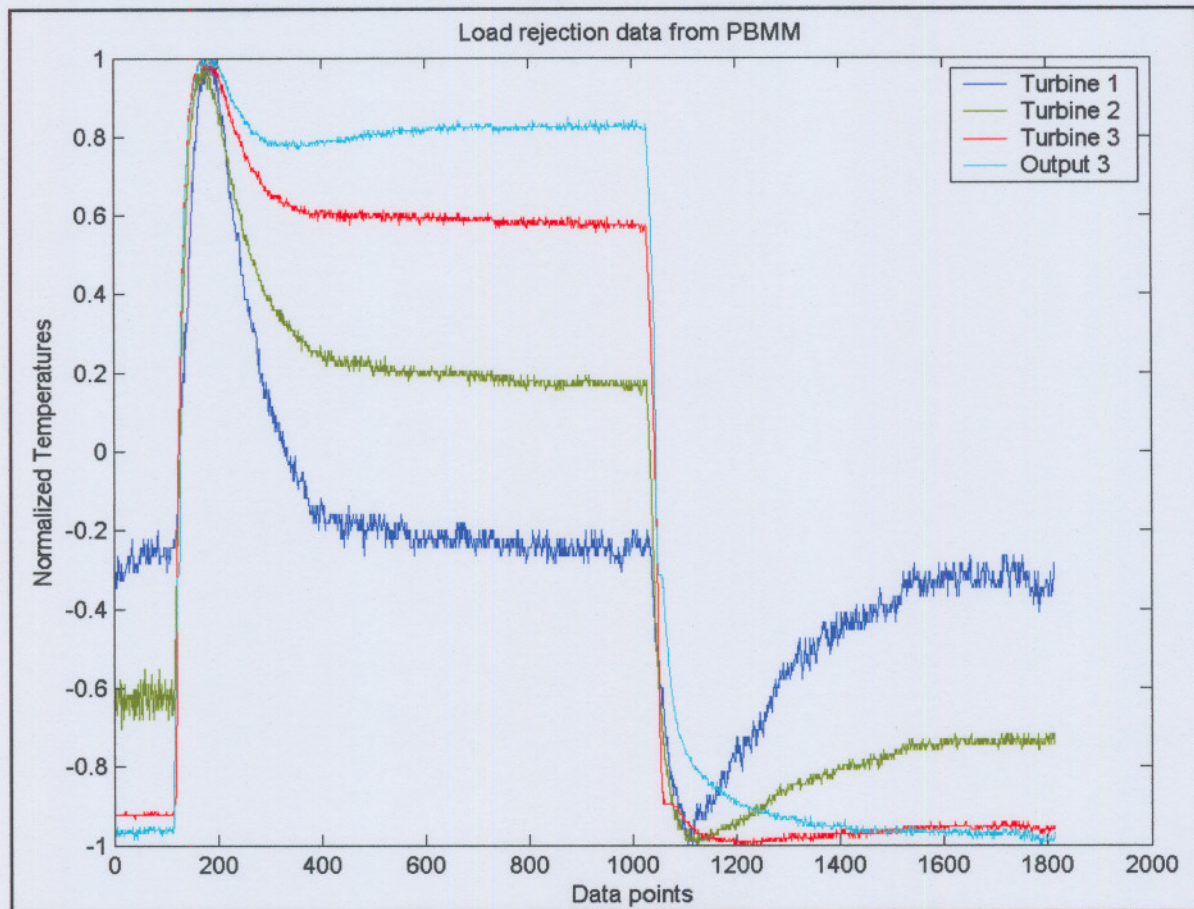


Figure 6.18: Load rejection test of the PBMM.

This is also input/output data for turbines 1, 2 and 3 as defined in the earlier sections. These turbines represent in the same order the high-pressure turbine, low-pressure turbine and the power turbine in the Brayton cycle.

FLOWNET – BRAYTON CYCLE

Three neural networks were made with a 2-1 topology, each with 90 input delays. The data illustrated above was taken as training data. The MSE of each network after only a single run of a 1000 epochs, is as follows: turbine 1 had an error of 9.27×10^{-5} , turbine 2 an error of 0.000113 and finally turbine 3 an error of 0.000160. This example shows that the neural networks can be trained with both simulated and measured data. Due to the serious consequences that could result from faults, no actual hardware faults could be initiated for fault detection and diagnosis.

6.4 Conclusion

This final section showed that when accurate simulation software is available, it can be used to generate fault scenarios without affecting plant operation or causing physical damage to the plant. Further remarks are given in chapter 7.

CONCLUSION

7 CONCLUSION

A conclusion on the research in this paper is given along with some final remarks.

CONCLUSION

7.1 Conclusion

Definite goals were stated and referred to throughout the thesis to stay on track. These goals were the specific problems as stated in section 1.4:

- diagnosing faults by using neural networks to classify residuals;
- modelling of a process using neural networks for fault detection;
- integrating the above-mentioned solutions; and
- successfully implementing the FDD concept on the simulated pebble bed micro model (PBMM).

The classification of residuals was thoroughly investigated in chapter 4. Static neural networks were used, which means networks without delays in their inputs. These static multilayer feed-forward networks performed very well and even when noise were added and the data were quantised, showed good generalising abilities. The biggest problem in this chapter was when similar residuals were misclassified and suggested that mathematical or statistical manipulation of the data was needed. The overall results showed that this concept could work.

In chapter 5 neural networks were investigated for their modelling abilities. The non-linear response of the induction motor was a well-defined benchmark for this investigation. The results proved that neural networks are excellent modelling tools and the FDD concept was taken to the final two goals.

Throughout this thesis results showed that neural networks are capable of solving non-linear problems. The final chapter integrates all these capabilities and the feasibility of the integrated FDD concepts was investigated. Although the accuracy for the fault detection and classification was above 90%, there is still room for improvement. The depth of this investigation must be kept realistic. This investigation most certainly *points* to a feasible concept.

This thesis states that it investigates predictive maintenance and that it is not just sensors that become faulty. However when a process fault develops, sensor measurements are used to investigate these faults. Deviations of the process from the simulating neural network might have noisy characteristics rather than a steady value. Sensor faults will easily be distinguished from process faults.

CONCLUSION

A process fault would affect the whole sensor array and not only temperature or pressure or mass flow rate, but a combination of all the characteristic values that are measured. This thesis investigates the concept that, if deemed feasible, could be investigated more intensively, requiring domain knowledge and experience.

There is no doubt that this investigation highlights the idea that neural networks are one of the greatest modern tools that engineers have in their library. MATLAB® and SIMULINK® as used in this research are examples of such software tools.

This exciting work can be extended and implemented in future research projects. It will also serve as an indication, for other research in this area, as to which concepts could produce a feasible solution. This investigation highlights tendencies that will be invaluable for similar research.

Further research could look into aspects such as missing data from the time series such as due to computer network outages. Partial and full models of large systems could be investigated for sensitivity with respect to various parameter changes.

APPENDIX

8 APPENDIX

Experimental data that is too lengthy to put in the chapters are listed below.

APPENDIX

Performance data of classification neural networks

Architecture	Time (s)	Cum. Error	Architecture	Time (s)	Cum. Error	Architecture	Time (s)	Cum. Error
3 3	1223	2.371	14 3	734	0.4227	6 3 (2)	1050	0.0322
	1179	0.952		743	0.2266		1058	0.0414
	1245	1.066		759	0.1339		1031	0.0337
	1189	1.4927		740	0.1548		1079	0.047
	1108	0.746		766	0.1969		989	0.34
				745	0.1104		1048	0.117
				744	0.1308		1058	0.0322
				757	0.2567		1059	0.0162
				750	0.2278			
				744	0.4277			
Average	1188.8	1.32554		748.2	0.22883		1046.5	0.0824625
6 3	962	0.1679	18 3	675	0.33	24 3 (2)	724	0.1832
	918	0.1515		706	0.7109		731	0.3251
	967	0.0477		704	0.1873		729	0.2
	910	0.0498		720	0.3139		741	0.6277
	889	0.0874		713	0.3303		711	0.55
	874	0.0832		690	0.1167		734	0.1648
	843	0.0525		700	0.1333		717	0.3988
	838	0.3259		685	0.1197		742	0.3256
	892	0.0552		689	0.1556			
	852	0.0313		709	0.2859			
Average	894.5	0.10524		699.1	0.26836		728.625	0.3469
9 3	921	0.2048	24 3	628	0.1824			
	865	0.1325		634	0.3258			
	880	0.0999		632	0.2095			
	816	0.1844		644	0.6337			
	818	0.1872		619	0.5635			
	823	0.094		639	0.1767			
	873	0.2506		624	0.4058			
	807	0.0712		642	0.3449			
	804	0.0998		642	0.3749			
	846	0.1018		650	0.3576			
Average	845.3	0.14262		635.4	0.35748			

Table A.1: Performance data for the 3-network topology.

The two sets on the most right-hand column have a 2 in brackets besides the architecture. This is where the most accurate and fastest networks were compared. They were trained with 1050 epochs and the rest was trained with only 300. The outcome was clear enough after just 8 runs of the training.

APPENDIX

Architecture	Time (s)	Cum. Error	Architecture	Time (s)	Cum. Error
9 7	1442	1.0887	25 7	1027	1.2016
	1386	2.4609		924	0.7981
	1469	2.2707		988	2.1742
	1451	3.5443		1007	1.7546
	1447	2.2074		985	1.729
	1455	1.427		1035	2.992
	1509	1.6515		988	1.8656
	1374	1.173		902	1.4342
Average	1441.625	1.9779375		982	1.7436625
15 7	1163	2.087	30 7	788	2.0114
	1219	1.692		853	1.273
	1088	0.7837		919	2.0123
	1282	1.0865		850	1.2545
	1188	1.2833		922	1.7118
	1229	1.2097		832	0.8926
	1185	0.5311		800	0.8972
	1163	2.0876		841	1.571
Average	1189.625	1.3451125		850.625	1.452975
18 7	1095	0.829			
	1210	0.8927			
	1189	1.1999			
	1081	1.749			
	1177	1.0454			
	1111	1.251			
	1198	0.7865			
	1172	1.4346			
Average	1154.125	1.1485125			

Table A.2: Performance data for the single-network 2-layer architecture.

The networks in the above table were trained with 1500 epochs. The networks in table A.3 were trained with the same number of epochs.

APPENDIX

Architecture	Time (s)	Cum. Error	Architecture	Time (s)	Cum. Error	Architecture	Time (s)	Cum. Error
7 7 7	2302	1.987	12 7 7	2199	1.1303	15 9 7	2071	0.915
	2182	1.1201		2323	1.12765		2097	0.5624
	2232	1.0352		2105	0.525		2262	0.7761
	2171	1.405		2183	0.909		2142	1.019
	2225	0.7607		2074	0.4946		1966	0.533
	1970	0.511		2124	0.7996		1886	0.7562
	2056	0.7855		2012	1.0027		1958	0.5687
	2268	1.7855		2189	0.3632		1951	0.6046
	2052	0.9851		2142	0.3454		1869	0.7347
	2173	0.8177		2009	0.3263		1883	0.4254
Average	2163.1	1.11928		2136	0.702375		2008.5	0.68951
9 7 7	2241	0.883	12 9 7	2015	0.7402	15 12 7	1711	0.8791
	2147	1.328		1804	0.7764		1687	0.5207
	2078	0.8642		2116	0.5365		1611	0.4317
	1874	0.4624		2187	1.1717		1791	0.4278
	2186	1.6565		2126	1.0593		1882	0.443
	2265	0.881		1912	0.804		1715	0.784
	2270	1.164		1830	0.2593		1717	0.4894
	2296	0.8178		1966	1.1678		1760	0.31
	2076	0.4914		2018	0.3614		1678	0.5378
	2270	1.5394		1891	0.8488		1643	0.2543
Average	2170.3	1.00877		1986.5	0.77254		1719.5	0.50778
9 9 7	1887	0.5437	12 12 7	1568	0.7024	12 15 7	1646	0.7208
	1792	0.5706		1811	0.7284		1528	0.9452
	1976	1.249		1885	0.6276		1468	0.452
	1946	0.8464		1791	0.6783		1532	0.5023
	1959	0.5977		1871	0.8544		1579	0.5866
	1868	0.756		1708	0.5665		1422	0.4192
	1877	0.4191		1636	0.5553		1558	0.7638
	1607	0.3938		1759	0.6657		1477	0.5367
	1838	0.4605		1657	0.2965		1585	0.4651
	1492	0.6909		1610	0.506		1657	1.0697
Average	1824.2	0.65277		1729.6	0.61811		1545.2	0.64614
15 15 7	1505	0.5451	7 12 7	1393	0.5925	12 18 7	1449	0.4149
	1678	0.5525		1668	0.4164		1353	1.2637
	1514	0.4542		1701	0.6691		1343	0.5881
	1493	0.8112		1582	0.8019		1419	0.5387
	1572	0.7703		1736	1.7163		1373	0.3472
	1548	0.6255		1655	1.0487		1336	0.4323
	1492	0.6209		1696	1.4583		1435	0.3239
	1512	0.7205		1522	0.3989		1387	0.5318
	1416	0.5929		1463	0.4746		1365	0.4378
	1555	0.4543		1644	0.6441		1477	0.7904
Average	1528.5	0.61474		1606	0.82208		1393.7	0.56688
9 12 7	1679	0.7211	7 15 7	1292	0.6086	15 18 7	1342	0.7011
	1597	0.2244		1315	0.9007		1564	0.5866
	1644	0.6613		1473	0.3623		1510	0.7891

APPENDIX

Architecture	Time (s)	Cum. Error	Architecture	Time (s)	Cum. Error	Architecture	Time (s)	Cum. Error
9 12 7	1611	0.8711	7 15 7	1419	0.6656	15 18 7	1526	0.5646
	1447	0.3318		1627	0.8573		1441	0.8352
	1749	1.126		1389	0.6649		1400	0.8471
	1722	0.5976		1557	1.0586		1426	1.081
	1568	0.7211		1589	0.3172		1507	0.9057
	1597	0.2244		1367	0.3208		1512	0.5519
	1646	0.6613		1471	1.01		1489	0.7132
Average	1626	0.61401		1449.9	0.6766		1471.7	0.75755
9 15 7	1356	0.4307	9 18 7	1388	0.9391			
	1522	0.4403		1410	0.9391			
	1586	0.5694		1314	0.6989			
	1460	0.9316		1265	0.4011			
	1390	0.7545		1259	0.8043			
	1697	0.3686		1410	0.4519			
	1532	0.6784		1352	1.1614			
	1482	0.5963		1374	0.7982			
	1581	0.6499		1484	1.6802			
	1507	0.6441		1523	0.6945			
Average	1511.3	0.60638		1377.9	0.85687			

Table A.3: Performance data for the single-network 3-layer architecture.

SNR(dB)	36	30	24	18.5	16.5	13
1	0.0166	0.0291	0.1614	0.2207	0.1638	0.3188
2	0.0197	0.0236	0.0327	0.1254	0.3483	0.4526
3	0.0191	0.0269	0.2419	0.2448	0.1584	0.541
4	0.018	0.0179	0.0674	0.2883	0.1865	0.3868
5	0.0161	0.0335	0.1287	0.0916	0.2786	0.2693
6	0.0162	0.0149	0.0425	0.425	0.0863	0.8173
7	0.0204	0.0335	0.0319	0.1502	0.3216	0.4704
8	0.0182	0.0231	0.1062	0.1102	0.2444	0.652
9	0.0148	0.0349	0.0289	0.0904	0.1923	0.5323
10	0.017	0.0187	0.0346	0.3562	0.1336	0.6822
Average	0.01761	0.02561	0.08762	0.21028	0.21138	0.51227

Table A.4: Performance with noise added for the three-network 6-3 architecture.

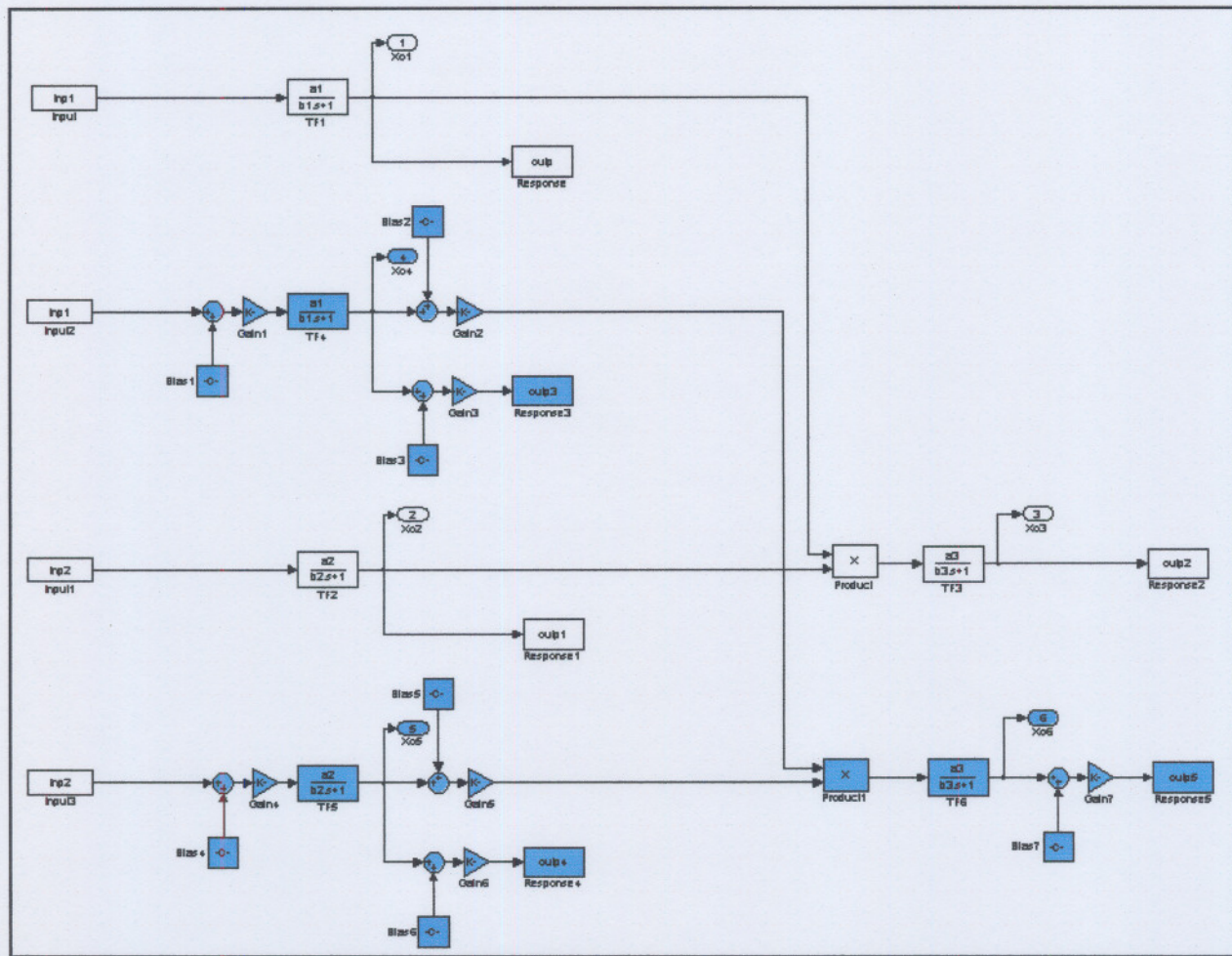
MISO model schematic illustration

Figure A.1: Schematic illustration of the multi-input single output model.

REFERENCES

- BARRINGER, H.P. & WOODROW, W.T. 2002. Process reliability: do you have it? - what's it worth to your plant to get it. Presentation at the 2002 AIChE National Spring Meeting in March 10-14, 2002, New Orleans, LA. Paper: #39f.
- BASSEVILLE, M. 1999. On fault detectability and isolability. IRISA Publication Interne. nr. 1240.
- CHEN, Y.M. & LEE, M.L. 2002. Neural networks-based scheme for system failure detection and diagnosis. *Mathematics and computers in simulation*. vol 58: 101-109.
- DASH, S. & VENKATASUBRAMANIAN, V. 2000. Challenges in the industrial applications of fault diagnostic systems. *Computers and chemical engineering*. vol 24: 785-791.
- BASSEVILLE, M. 1999. On fault detectability and Isolability. Institut de Regherche en Informatique et Systèmes Aléatoires. Publication Interne. nr. 1240.
- DUMMERMUTH, E. 1998. Advanced diagnostic methods in process control. *ISA Transactions*. vol 37: 79-85.
- FUENTE, M.J. & VEGA, P. 1999. Neural networks applied to fault detection of a biotechnological process. *Engineering applications in artificial intelligence*. vol 12: 569-584.
- HAYKIN, S. 1999. Neural Networks: a comprehensive foundation, 2nd ed. N.J.: Prentice Hall.
- HUSH, D.R., ABDALLAH, C.T., HEILEMAN, G.L., & DOCAMPO, D. Neural networks in fault detection: A case study. EECE Department, University of New Mexico, Albuquerque, NM 87131, USA. <http://www.eece.unm.edu/controls/papers/ACC97-paper1.pdf> [Date of access: 19 June 2003].
- JAZWINSKI, A.H. 1970. Stochastic processes and filtering theory, N.Y: Academic Press.

LEE, S. & KIL, R.M. 1991. A Gaussian potential function network with hierarchically self-organizing learning, *Neural Networks*. vol 4: 207–224.

MOODY, J. & DARKEN, C.J. 1989. Fast learning in network of locally-tuned processing units. *Neural computation*. vol 1: 281–294.

MUNOZ, A. & SANZ-BOBI, M.A. 1998. An incipient fault detection system based on the probabilistic radial basis function network: application to the diagnosis of the condenser of a coal power plant. *Neurocomputing*. vol 23: 177-194.

ONG, C.M. 1998. Dynamic Simulation of Electric Machinery using Matlab/Simulink, Prentice Hall PTR.

RENGASWAMY, R. & VENKATASUBRMANIAN, V. 2000. A fast training neural network and its updaton for incipient fault detection and diagnosis. *Computers and chemical engineering*. vol 24: 431-437.

SIMANI, S. & FANTUZZI, C. 2000. Fault diagnosis in power plant using neural networks. *Information sciences*. vol 127: 125-136.

The PBMR webpage. <http://www.pbmr.co.za>. [Date of access: 30 September 2002]

University of Washington faculty web server. <http://faculty.washington.edu/chudler/cells.html>. [Date of access: 20 February 2003]

University of Sheffield webpage. <http://www.shef.ac.uk/materials/SSM/imi/squicage.jpg>.

The Flownet webpage. http://www.flownet.net/frameset_default.html.