Issues in Verification and Validation of Neural Network Based Approaches for Fault-Diagnosis in Autonomous Systems

Uma Bharathi Ramachandran

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science in
Electrical and Computer Engineering at
Concordia University
Montreal, Quebec, Canada

September 2005

Your file Votre référence
ISBN: 0-494-10249-7
Our file Notre référence
ISBN: 0-494-10249-7

Abstract

Issues in verification and validation of artificial neural network based approaches for fault diagnosis in autonomous systems

Uma Bharathi Ramachandran

Autonomous systems are those that evolve over time, and through learning, can make intelligent decisions when faced with unidentified and unknown situations. Artificial Neural Networks (ANN) has been applied to an increasing number of real-world problems with considerable complexity. Due to their learning abilities, ANN-based systems have been increasingly attracting attention in applications where autonomy is critical and where identification of possible fault scenarios is not exhaustive before hand.

The "black box" label associated with ANNs hides away training rules by which they have adapted to arrive at the expected result. This notion of unpredictability of neural networks makes it hard to warrant a certification procedure and thereby limits their use in safety critical systems. There are no effective ways to determine (a) if the network has learned the correct data even if the training set provided all possible scenarios, and also (b) how the network would adapt when it is faced with unfamiliar data that it has not seen during the training phase. This necessitates a need for well-tested techniques for Verification and Validation (V&V) of neural networks in order to verify the correctness and robustness of the rules that the network has learned. There are a few proposed solutions to V&V of autonomous systems, specially the ones with ANN-based. However

the solutions are far from being matured and complete. These challenges are still open issues in this domain.

Through extensive literature review it was established that ANN based systems cannot be verified by using conventional V&V techniques due to their own inherent limitations. Verification of trained neural networks does not allow us to directly prove that the network behaves according to a certain specification. Thus there is an immediate need for a formal methodology to verify the correctness of the learning rules. We envision that formal methods can be used successfully for verifying neural network based software systems.

Towards this end, we have proposed a methodology in which the learning rules that a trained network has adapted can be extracted and refined using rule extraction and rule refinement techniques, respectively, and then these refined rules are subsequently formally specified and verified against requirements specification using formal methods. The effectiveness of the proposed approach has been demonstrated using a case study of an attitude control subsystem of a satellite.

Acknowledgements

I would like to take this opportunity to thank my supervisors Dr. Purnendu Sinha and Dr. Khorasani for their guidance in carrying out this research. While Dr. Sinha has helped me to understand what research is all about, Dr. Khorasani has challenged me with difficult questions at different stages of this work which has guided me in my research. My sincere thanks also go to Mr. Karthik Vijayakumar who has provided me with valuable suggestions at different stages of this thesis. I would like to express my appreciation to the examiners for their valuable comments for improving the presentation of this thesis.

My parents and siblings have always been the key factor behind every success in my life. I would not be able to finish this Masters degree without their encouragement. I would like to express my gratitude to my friends who have always been an inspiration. My sincere thanks also go to Dr.Rama Bhat and family for all the help and encouragement. He and his family have been the reason why I am doing this course now at Concordia University.

My friends in Montreal – especially, my roommates, my Research Group friends, Ashish Tiwari and Ekta Khurana – have extended their helping hands towards me whenever I needed it and have always driven me to success and taught me the meaning of perseverance. They have also been the sources of lots of fun in my life. All of them have been there whenever I needed their support and encouragement and have been more than family to me here in Montréal.

Table of Contents

List of Figures ix

List of Tables xi

List of Abbreviations xii

1 Introduction

1.1 Overview of Adaptive and Autonomous Systems 1

1.2 Neural Network in Safety Critical Applications 2

1.3 Need for Formal Verification of Neural Network based Software Systems 3

1.4 Issues in Verification and Validation of Neural Network based Software Systems 5

1.5 Approaches to V&V of Neural Network based Software Systems 8

1.6 Observation and Motivation 10

1.7 Contributions of the thesis 11

1.8 Organization 12

2 Literature Review

2.1 Overview of Artificial Neural Networks 14

2.1.1 Data Representation in Neural Networks 14

2.1.2 Neural Network Architecture and Topologies 16

2.1.3 Training and Learning algorithms 17

2.1.3.1 Back propagation Learning 18

2.1.3.2 Other Learning Algorithm 21

2.2 Overview of safety critical systems 25

2.2.1 Basic definition 25

2.2.2 Software System Safety Analysis 26

2.2.3 Challenges for safety critical neural network 27

2.3 Conclusion 28

3 V&V of Neural Network: Overview and Proposed Approach.

3.1 An Overview of Rule Extraction 29

3.2 Rule Extractions Algorithms 31

3.2.1 Boolean Rule extraction using Decompositional Approach 32

3.2.2 Extraction of Fuzzy rules 35

3.2.3 Boolean Rule extraction using Pedagogical Approach 35

3.3 Need to develop Fault Diagnosis Systems 37

3.3.1 Development of ANN based Fault Diagnosis Systems 39

3.3.2 Need for verification of ANN based Fault Diagnosis System 42

3.4 Proposed Approach: Overview 43

3.5 Formal verification 49

3.6 UPPAAL: Overview 50

3.6.1 Modeling in UPPAAL 50

3.6.2 Simulation and Model checking 51

3.7 PVS: Overview 52

3.8 Conclusion 53

4. Verification of the ACS model

4.1 ACS Failure 54

4.1.1 ACS Failure Scenarios -1 55

4.1.2 ACS Failure Scenarios -2 56

4.1.3 ACS Failure Scenarios -3 57

4.2 Neural Network for the ACS Simulink Model 59

4.2.1 Preprocessing and Post processing 63

4.2.2 Neural Network Training using the Simulation Results 64

4.3 Rule Extraction Algorithm applied to Trained Neural network 73

4.3.1 Rule Extraction using SUBSET Algorithm 75

4.3.2 Rule Extraction using VIA Algorithm 78

4.4 Formal Specification of the neural networks in UPPAAL 88

4.4.1 Simulation and Verification 92

4.5 Formal Specification in PVS 95

4.6 Conclusion 98

5 Conclusions and Future work 101

Bibliography 106

List of Figures

2.1	Structure of Neuron in a Neural Network	15
2.2	Threshold (L), hyperbolic tangent(C) and Sigmoid(R) activation functions for a neuron	16
2.3	Back propagation structure	19
2.4	Perceptron attempting to express the XOR problem	22
2.5	The XOR problem showing that no hyperplane can divide the given set of points into two classes	23
3.1	Different bodies that influence safety critical systems	38
3.2	Domino model used in *PROforma*	40
3.3	Block Diagram of the Proposed Approach	43
3.4	ACS Block Diagram	44
3.5	Framework of combining the symbolic knowledge and neural network learning	46
4.1	Pitch error vs time under failure scenario-1	54
4.2	Pitch error vs time under failure scenario-2	55
4.3	Pitch error vs time under failure scenario-3	56
4.4	Different ACS Parameters under failure scenario-3	57
4.5	A multilayer perceptron architecture for the model chosen for training	58
4.6	Performing early training stopping to ensure best generalizations	61

4.7	Graphical representation of error in training (motor current)	65
4.8	Graphical representation of correct training (motor current)	65
4.9	ANN versus simulation data	67
4.10	Graphical representation of error in training (Bus Voltage)	68
4.11	Graphical representation of correct training (Bus Voltage)	69
4.12	ANN versus simulation data	69
4.13	Graph showing the correct training (Pitch)	70
4.14	Graph showing the error in training (Pitch)	71
4.15	ANN versus simulation data	71
4.16	Diagram showing initial conditions from which rules are extracted	76
4.17	Network topology of the Neuron model under Consideration	79
4.18	Directed acrylic graph for rules with three Boolean variables	85
4.19	Main finite state machine in UPPAAL	88
4.20	Inputlayer finite state machines in UPPAAL	89
4.21	Hiddenlayer finite state machines in UPPAAL	90
4.22	Outputlayer finite state machine in UPPAAL	90
4.23	Simulation Results	92

List of Tables

2.1	Truth Table of XOR	22
4.1	Threshold values for data variable considered as the inputs to the ANN	62
4.2	ANN parameters for the ACS model.	64
4.3	Testing the ANN with 10 sets of training	66
4.4	Testing the ANN after 10 sets of training	68
4.5	Testing the ANN with 10 sets of training	70
4.6	Weights classified in descending order for positive and negative weights	75
4.7	Weights and bias of the neural model that was shown in figure 4.17	79
4.8	The extracted rules from the network learned	84
4.9	Classification and applicability	86
4.10	Quality of rules and complexity	87
4.11	Model checking and reachability analysis of the system in UPPAAL	94

List of Abbreviations

1	ANN	Artificial Neural Networks
2	V&V	Verification and Validation
3	MLP	Multi-layer Perceptron
4	AI	Artificial Intelligence
5	VIA	Validity Interval Analysis
6	ACS	Attitude Control Subsystem
7	ACP	Attitude Control Processor
8	RW	Reaction Wheel
9	MW	Momentum Wheel
10	MTB	Magnetic Torque Bar
11	PVS	Prototype Verification System
12	SRI	Stanford Research Labs
13	DAG	Directed Acrylic Graph

Chapter 1

Introduction

This chapter begins with a brief introduction to adaptive and autonomous systems and their role in safety critical applications. It further highlights the need for verification of neural network based software systems which are adaptive systems used in safety critical applications. The discussion then leads to the issues which arise when verifying and validating Artificial Neural Networks (ANN) based software. These issues form the basis for our proposed approach for Verification and Validation (V&V) of ANN-based autonomous systems. The chapter concludes with highlighting, observations, and contributions made in this thesis.

1.1 An Overview of Adaptive and Autonomous Systems

Autonomous systems are composed of several technical devices. These devices have both onboard intelligence as well as certain standalone communication capabilities. The flexibility of these devices has allowed these autonomous systems to grow in a short time span and become a very significant part of the present market.

Adaptive systems are those which have the ability to adapt to changing environments. They are being widely used in domains where it is impossible to predetermine all the possible environmental conditions that may arise. These systems, however, are hard to model as they have different issues such as many degrees of freedom, distributed sensors, high noise level, and uncertainty. Nevertheless, the intelligence of these adaptive systems is measured with respect to their ability to handle

the difficulties present within the system.

In adaptive systems, such as unmanned space vehicles, the focus of autonomy is likely to be vital to their functionality. Allowing the system to diagnose and fix unexpected problems onboard is a valuable technology for unmanned aircraft and space vehicles. These autonomous systems help in reducing the ground support required and the effect of communication lag between the ground station and the space vehicle. This adaptive behavior assures that the system will perform reliably while minimizing resources utility. These adaptive systems should be able to intelligently respond to changes in the system and execute efficient resource allocation. This implies that adaptive systems should contain the capability of assuring fault tolerance. Thus adaptive systems are those which evolve over time and through learning, can make intelligent decisions when faced with unidentified and unknown situations.

Artificial Neural Networks (ANN)-based systems have emerged as a powerful class of autonomous and adaptive systems [1]. NASA has conducted experiments using adaptive computational paradigms, such as ANN for providing fault tolerance capabilities in control systems where sensor or actuator faults are present. Promising results [1] have provided ways to future work. For this reason, ANN-based autonomous systems are being considered as a promising solution to the autonomy problem owing to the distance in space mission applications.

1.2 Neural Networks in Safety Critical Applications

Artificial Neural Network (ANN) based information-processing paradigms belong to a special class of autonomous systems. They have been applied to an increasing

number of real-world problems with considerable complexity. ANN is different from conventional computing as they are self-organized and adaptive, while conventional computing use an algorithmic approach to a problem. Due to their learning abilities, ANNs have been increasingly attracting attention in applications where autonomy is critical and where the identification of the possible fault scenarios is not extensive before hand. The computational capacity of ANN excels in many areas:

- Pattern classification
- Speech analysis
- Robot steering
- Processing of inaccurate or incomplete inputs

Further, some of the abilities of ANN that should be explicitly stated include error tolerance, ability to deal with novel inputs, ability to work with problems where algorithmic specifications are not available, and speed. Learning algorithms indirectly aid the neural network in accomplishing these abilities.

1.3 Need for Formal Verification of Neural Network based Software Systems

Though ANN have been accepted and used in fault diagnosis, there are some specific concerns with respect to the reliability and acceptability of ANN in safety critical applications, since faults in the system may lead to catastrophic effects. The major shortcoming that we have noticed when it comes to ANN is the lack of human comprehensibility. Hence, the correctness checking of ANN based systems is important

before applying them to safety critical applications.

Verification is a process of checking if the system conforms to the specification, which is to assure that the software components or software systems meet their specified requirements [2]. Specified requirements are not only found in requirements-specification documents, but can also be found in functional specifications, architecture and design models, test cases, and so on. The purpose of validation is to demonstrate that a software component or system fulfills its usage requirements when placed in its intended environment [3].

Although effective, applying verification and validation to neural network-based systems, however, is difficult since the underlying learning algorithm does not provide any mathematical contribution in the verification process. Further, the black box label restricts us from determining the learning rule that the network adapted to arrive at the specific training result. As a result, many authors have attempted to find solutions to this problem by extracting the knowledge stored in connection weights and interpreting the activation of each and every neuron. This knowledge reveals some information about the black box and then rules can be extracted from the ANN.

As a consequence, there is no defined way to determine if the network has learned the correct data even if the training set provides all possible situations of scenario under test. It is also not clear if the network would adapt well when faced with unfamiliar and unseen data during the training phase. This introduces a need for well-tested techniques for the verification of ANN in order to verify the rules that the network has learned. Furthermore, such verification is needed to see how the network is adapting and to determine if the network is in fact learning properly.

ANNs are adaptive systems and thus, they are supposed to adapt to unforeseen situations, but due to the lack of a detailed system model, traditional verification techniques are meaningless. Thus, formal techniques must be devised for the Verification and Validation (V&V) of neural networks. However, to verify ANN-based systems with the formal verification techniques, there is a need to have a detailed system model [3]. This requirement is set forth to focus on the risk associated with the technical development and the performance of ANN. The major prerequisite for the certification procedure is that all activities in the software development lifecycle have to be executed according to the prescribed detail software process. Essentially, formal methods are necessary for making V&V of software more objective by complimenting the traditional testing technique. They are required to be more rigorous such that there is an increased assurance with respect to the results of the conducted V&V process. Formal methods for ANN include specification language and methods for knowledge representation, such as rule extraction and decision trees. These formal methods can allow the system to be tested for both correctness and completeness.

1.4 Issues in Verification and Validation of Neural Network based Software Systems

Some of the well-established techniques to verify conventional software systems are [4]:

1. Model checking
2. Theorem proving
3. Static analysis
4. Run time analysis

However, ANN based software systems cannot be verified as conventional software systems. Conventional software systems can be verified against a model but ANN based systems lack a system model therefore they cannot be compared in the verification phase.

In the V&V of software systems, we consider three factors, namely, fault avoidance, fault removal, and fault tolerance [4]. Unfortunately, none of these three factors would be applicable to verifying the ANN for the following reasons.

Fault Avoidance:

Fault avoidance does not hold good for the adaptive ANN, as we do not have the knowledge of how these systems learn the rules and we are unaware of what they would learn. There is no bound on what the system is learning as there would be an unreliability always encountered with the ANN.

Fault Removal:

Fault removal also does not hold good for adaptive ANN, as we cannot anticipate the type of faults. Within the context of conventional software system, the assumption is that the system of interest will work in the same way in the real world as it had worked during the training phase. However, with respect to ANN, we cannot assure that the system will repeat its functionalities for the training and real world phases when given the same input data. The behaviour of the ANN is based on the learning rule and we are not aware of the reaction of the system when it comes across the data which had been seen during the training phase and thus fault removal property doesn't hold true.

Fault Tolerance:

Fault tolerance works on the premise that we know exactly the kind of faults to

anticipate and we tolerate those by using techniques, such as error detection and error recovery. This is not possible when working with ANN-based autonomous systems, as there is no way to anticipate neither the type of faults nor the techniques by which we know what the system has learned.

The V&V of conventional software system begins with a system model and has been tested against this neural network model. Techniques like *run time verification* have some scope for use in V&V of neural network-based systems. However, even this technique has some drawbacks which include overhead in program execution and the possibility of falsely detecting a non-existent problem. The uncertainty associated with non-deterministic software systems such as ANN, makes the verification process difficult. For safety-critical applications, a neural network-based controller must be verified and validated thoroughly, and must pass a rigorous certification procedure which has yet to be accomplished. There are a number of questions raised about the capabilities of ANN, such as their memory retention capability, their abilities to learn the correct data, handle data outside the training set, and adapt to unseen scenarios [4].

Experts have to know what information they are looking for and how to analyze the data. Previously lookup tables were used to analyze the mapping between the independent variables and the dependent variables. However lookup tables can only be used when the mapping is comparatively simpler. Static ANN, for example, behaves like lookup tables. Other complexities that arise when we use continuous input variables are that infinite input values may be present and it is not feasible to check every possible output. Since ANN is non-linear, it is difficult to guarantee the bound on the interior values. Thus, as the output set becomes large, it becomes more difficult to guarantee any

kind of proper data analysis. Performance guarantee is essential for ANN to be able to clear the certification procedure and make their way into the class of safety critical applications. These challenges are still open issues in the field of V&V of ANN.

1.5 Approaches to V&V of Neural Network based Software Systems

During the course of our literature survey, we found that for ANN-based systems to be verified by the conventional V&V techniques there are different techniques that have been adapted. One such technique discusses the detailed development process for the neural network which has not yet been trained. The certification procedure requires that the entire software development and lifecycle be performed according to a software process. A process guide has been developed for the intelligent flight control software (IFCS) [5]. They have been specially modified for ANN. V&V of ANN was applied to all phases of the software lifecycle. These software development lifecycles are applied to pre-trained ANNs.

Some of the techniques that were discussed for post-trained neural network as possible solutions have been briefly mentioned here. The techniques that claim to address and resolve some of the V&V issues in ANN include [6, 7, 8]:

- Automated testing and test data generation methods
- Run-time monitoring
- Formal methods
- Cross validation
- Lyapunov Stability Analysis

- **Automated testing and test data generation methods**

The traditional training-validation-testing approach fails to give assurance that a neural network will meet the rigorous standards of safety critical applications. A testing set for error calculation is enough for a developer to assess a system since this technique does not hold well for highly reliable environments. So this technique enables the developer to increase the amount of testing data which aids in reliability assessment. Test data generators are used to generate partial or whole new data sets. This technique works well in situations where the data used is a single-valued data and where fixed neural networks are used. Hence, this technique helps the V&V practioners in the testing phase of the network.

- **Run-time monitoring**

Run-time monitors are developed as a part of the system itself. They need not be upgraded from the conventional testing technique. They can detect if the network is deviating from the learning curve. This technique is very useful as it helps in detecting faults. However, this technique has a few drawbacks which include overhead in program execution and a possibility of falsely detecting a non-existent problem.

- **Formal methods**

Formal methods-based technique includes specification language and methods for knowledge representation including rule extraction and decision tree. The use of a specification language such as CONNECT or nn/xnn for the abstract specification of the ANN has been discussed in detail in [5]. Furthermore, rule extraction and decision trees have been seen as a means of providing an insight into the rules that the network has adapted. The developed approach is based on these initial findings.

- **Cross Validation**

This is a technique where the system assures reliability through redundancy. The validation is in-built in such systems with cross validation, where the system uses a group of neural networks to validate one neural network. This technique improves system reliability and can also be applied in safety critical applications. This technique can be applied to the design phase of the development of the neural network. The drawback of this process is that it is time consuming and tedious.

- **Lyapunov's Stability Analysis**

This technique deals with self-stabilization analysis in the design, verification and validation of the dynamics of an adaptive system. Often, mathematical theory for stability analysis is (mis)understood as a process for solving differential equations that describe the system dynamics. Stability analysis is the theory of validating the existence of stable states. Since traditional self stabilization approaches cannot deal with continuous adaptation and lack flexibility, Lyapunov's second method for neural networks can be used to validate the existence of the stable state in the adaptive systems [8]. But when dealing with variable data manifold, we propose the need for a real-time stability monitor that can detect unstable state deviations.

After discussion of the related work done in this field, we would like to just state our observations, as well as, the contributions that were made in this thesis.

1.6 Observations and Motivations

There is an increasing demand to make adaptive systems safer and reliable. Their requirements extend beyond that are normally acceptable for non-critical software

systems. Adaptive systems evolve over time and this has been seen as their greatest advantage when used in safety critical (and autonomous) applications. The system has to be adaptive to recover from the faults or have early detection capabilities so that we can avoid shutdown, breakdown, or even catastrophes involving human life. Given these issues, verification and validation of adaptive systems become a necessity.

ANN belongs to a special class of these adaptive systems. It has been increasingly attracting attention in safety critical applications. Thus, the ANN used in critical applications have to be verified thoroughly for them to be accepted and used in safety critical applications. With this background, we have studied and addressed issues related to verification and validation of ANN used in safety critical applications.

1.7 Contributions of the Thesis

While conducting literature survey, it has been observed that the already-established frameworks for verification and validation of ANN present in open literature have not addressed formal verification of neural network systems [6]. Earlier work in this field had concentrated on the same problem but from a different aspect. They were trying to replace lookup tables with fixed neural network. The non-linear aspect of the fixed neural network was overcome by an estimation technique that was applied. This technique is limited to networks that are trained offline and is frozen after training.

On the other hand, we chose an autonomous model of a system to show verification and validation of ANN adapted at safety critical applications. These chosen ANN find their place in a special class of autonomous systems. Furthermore, space applications are taken into consideration, as distance is a major drawback in such

systems. Thus, increasing duties conducted at the ground station were made to be self-directing in the spacecraft itself as a means of overcoming the distance factor. This also allows fault diagnosis and recovery to take place autonomously in the spacecraft. Thus V&V of neural network-based software systems is essential within safety critical system, since they might lead to undesirable catastrophic effects if they are not tested for reliable performance.

Our specific contributions in this thesis are:

1) Address issues in the V&V of the ANN
2) Development of an approach for verification and validation of ANN that have been used in software systems as a part of larger software systems.
3) Demonstration of the same with an ACS model and training the neural network to provide fault tolerance for the ACS model.
4) Demonstrating rule extraction from trained neural network to enable comprehensibility of the neural network model developed.
5) Demonstrating the use of formal verification to facilitate neural networks to be used in safety critical systems.

1.8 Organization

The thesis is organized as follows: In Chapter 2, we give an introduction of ANN and definition of safety critical systems, as well as, the challenges in safety critical neural networks. Chapter 3 introduces rule extraction from a trained neural network and describes the different approaches involved. Further, it introduces the model used to illustrate the proposed approach. Chapter 4 describes three failure scenarios in the model

followed by the process of training the ANN and finally the rule extraction from trained neural network. The formal specification of the model is also described in this chapter. Finally, in chapter 5 we conclude with a brief discussion of contributions of the thesis and future research direction.

Chapter 2

Literature Review and Survey

The first part of this chapter gives an overview of ANN and the learning algorithms that have been used. This chapter further focuses on safety critical systems and the challenges of safety critical neural networks.

2.1 An Overview of Artificial Neural Networks

ANN is viewed differently from conventional computing. Neural networks are essentially information processing paradigms that have been greatly inspired by the biological nervous systems [9]. They are a form of multiprocessor computer systems with simple processing elements, highly interconnected processing elements (neurons), and adaptive nature. Moreover, they can be trained to perform a specific function.

2.1.1 Data Representation in Neural Networks

The ANN looks similar to the biological neural cells. They have numerous simple processing elements (*neurons*) which are interconnected. These neurons are responsible for transporting the incoming information on one neuron to the outgoing connections of the connecting neurons. These connections are called *weights*. Each of these weights has a specific value. Artificial neural networks work the same way as the human brain [9]. The information is sent to the neuron using these input weights. This input is processed by a propagation function that adds up the values of all incoming weights.

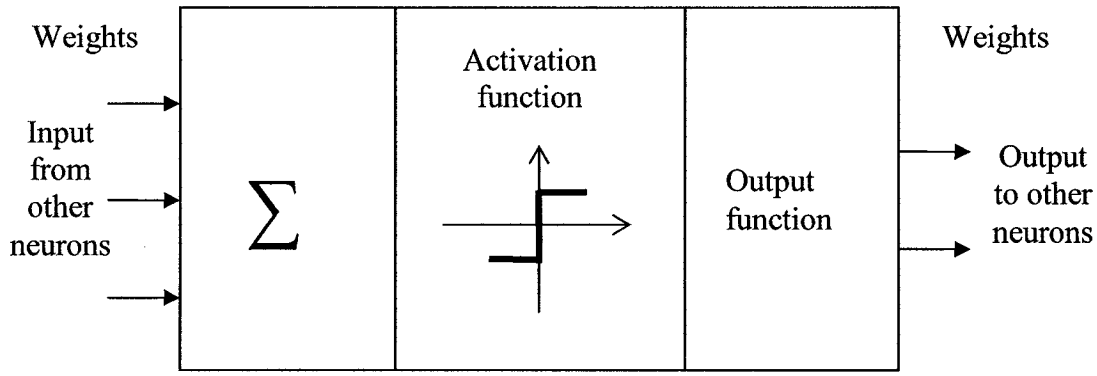


Figure 2.1 Structure of a neuron in a neural net [9].

The resulting value is compared with a threshold value by the neuron's activation function. If the neuron exceeds the threshold value, it is activated, else it is inhibited. If activated, the neuron sends an output on its outgoing weights to all connected neurons.

The neurons are usually structured in layers. The neurons in one layer are connected to a neuron in a preceding or following layer. This creates an interconnected neural net. The information given to a neural net is propagated layer-by-layer from input layer to output layer through none, one, or more hidden layers. The structure of the neuron in a neural network shown above in Figure 2.1, illustrates how this activation takes place. Figure 2.2 shows the different threshold functions that can be chosen for activation of the neuron.

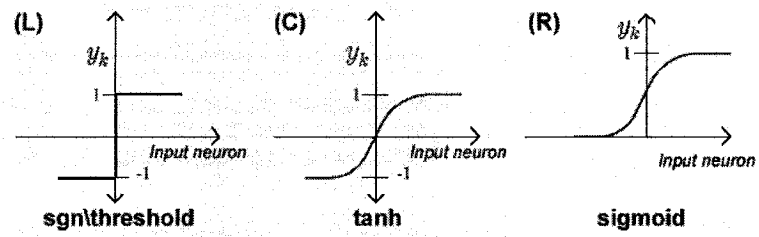


Figure 2.2 Threshold (L), hyperbolic tangent (C) and Sigmoid (R) activation functions for a neuron [9].

Depending on the learning algorithm, it is also possible that information is propagated backwards through the net.

2.1.2 Neural Network Architecture and Topologies

There are different types of ANN. The ANN can be classified as feed forward and feed back networks [9, 10].

Feed Forward Networks:

Feed forward networks are those which contain layers of neurons in which the input layer is connected to the output layer only in one direction. Hidden units perform the activation function and the outputs from the hidden layer either feed into another hidden layer (if multiple hidden layers exist) or the output layer.

Feed Back Networks:

Alternatively, there are feed back networks which are dynamic. Their 'state' continuously changes until they reach a stable point. They remain at the stable point until the input changes and a new stable point needs to be found. For this reason, they are very powerful, but can become very complex. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback

connections in single-layer organizations. To achieve the desired function, the network can be influenced by [9]:

- **The learning algorithm** – They are used to minimize the error between the actual output and the desired target function.
- **Number of hidden units** –More hidden units is appropriate for inputs with more noise.
- **The number of learning samples** – More training samples may provide more data for the target function to be achieved easily.

2.1.3 Training and Learning Algorithms

Once the neural network architecture has been designed, a training process and learning algorithm must be selected. This process is where the network is trained to perform a specific function by adjusting connection weights based on a learning rule. The training process consists of three sets of data: training data, testing data, and validation data. The training sample data is usually chosen to be a true representative of a target function. There are different algorithms that have been adopted which try and minimize the error in the target function as much as possible. The different learning algorithms that have been established are given below.

Supervised learning: Supervised learning is a type of learning algorithm, where the output of the network is compared with a target output. Depending on the differences between the input and the target function, the error is computed.

Unsupervised (self-organizing) learning: A specific type of a learning algorithm, especially for self-organizing neural networks like the Kohonen Feature Map. Unlike

supervised learning, no target patterns exist.

Reinforcement Learning: This can be seen as an alternative of the supervised learning technique. It is used when learning examples (inputs and outputs) are not available. Supervised and unsupervised learning are commonly used with the following weight updating methods:

Off-line learning: The network weight updates are computed and recorded during one pass of the training set. At the end of the training, all the weights are added together and the network weights are updated with the composite value.

On-line learning: The network weight updates are computed and modified after each input sample.

2.1.3.1 Backpropagation Learning

The Backpropagation Net was first introduced by G.E Hinton, E.Rumelhart, and R.J Williams, and is one of the most powerful neural net types [9]. It has a structure that is similar to a multi-layer-perceptron and uses the backpropagation algorithm. These are feed forward types of ANN which use supervised learning methods. Their structure is similar to the normal neural net types where they have an input layer, a few hidden layers and an output layer.

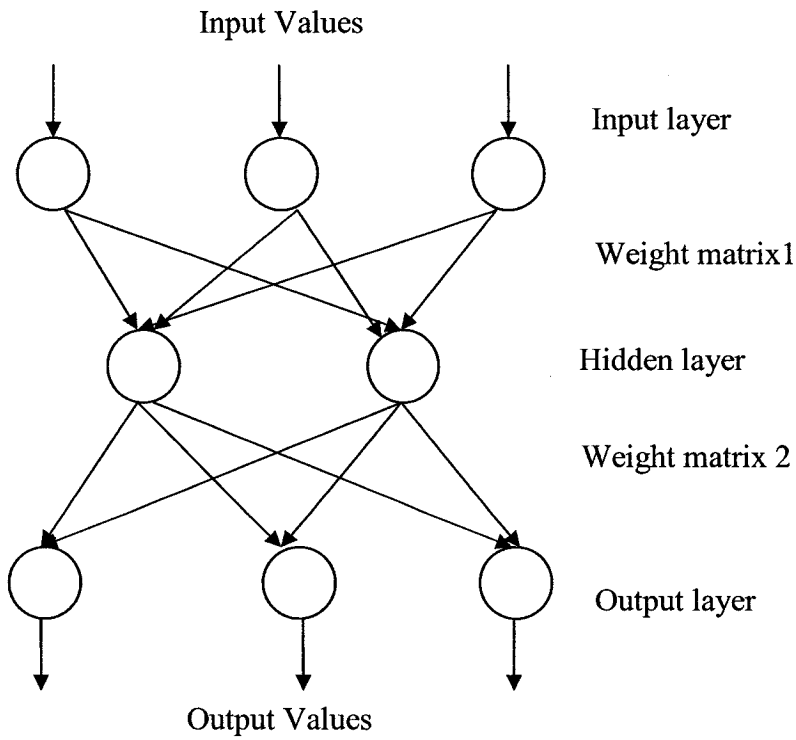


Figure 2.3 BackPropagation structure [9].

The BackPropagation algorithm uses a computed output error to change the weight values in the backward direction. To get this net error, a forward propagation phase must be applied. While propagating in forward direction, the neurons are being activated using the *sigmoid activation function.*

The formula of **sigmoid activation** is shown in equation 2.1:

$$f(x) = \frac{1}{1 + e^{(-x)}} \qquad (2.1)$$

Figure 2.3 shows the structure of the backpropogation neural network. Assuming

neurons in the model are arranged in such a way that the first neuron is called neuron (i) in the first layer and the second neuron in the first input layer is (i+1) and so on. The same process is repeated for the second layer where the second layer is defined as (j), (j+1) and so on. The following algorithm steps are applied in the back propagation:

1. Set all weights to random values ranging from -1.0 to +1.0.
2. Set an input pattern (binary values) to the neurons of the net's input layer.
3. Activate each neuron of the following layer:

    - Multiply the weight values of the connections leading to this neuron with the output values of the preceding neurons.
    - Add up these values.
    - Pass the result to an activation function, which computes the output value of this neuron.

4. Repeat this until the output layer is reached.
5. Compare the calculated output pattern to the desired target pattern and compute an error value.
6. Change all the weight values of each weight matrix using the formula :

    Weight (old) + learning rate * output error * output (neurons i) * output (neurons i+1) * (1 - output (neurons i+1))

    Where learning rate is defined as the changeable value used by the learning algorithm which affects the changing of the weight values.

7. Go to step 1.

8. The algorithm ends, if all output patterns match their target patterns.

2.1.3.2 Other Learning Algorithms

There are many other learning algorithms and network models. Variations of recurrent or feed-back networks influence the learning algorithms utilized. These networks contain connection cycles from the output layer back into the input layer. All learning rules derived for the multi-layered perceptron can be used for this type of network [9].

Perceptron

The perceptron was first introduced by F. Rosenblatt in 1958 [9]. It is a very simple neural net type with two neuron layers that accepts only binary input and output values (0 or 1). The learning process is supervised and the net is able to solve basic logical operations like AND and OR. It is also used for pattern classification purposes. They usually have one input layer and one output layer. They are feed forward type of networks and they use the Hebbian learning rule [9]. There are however some problems that cannot be solved by perceptron. The most often quoted problem is the *XOR problem* which involves building a perceptron, which takes two Boolean input and outputs the XOR of them. What we expect here is a perceptron which would output a 1 if the two inputs are equal and output a zero if the inputs are unequal. The truth table showing the same is shown in Table 2.1

Input		Desired Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1 Truth Table of XOR

Consider the sample perceptron shown in figure 2.4. If the inputs are both 0, then net input is 0, which is less than the threshold (0.5). So the output is 0 - desired output. If one of the inputs is 0 and the other is 1, then the net input is 1.

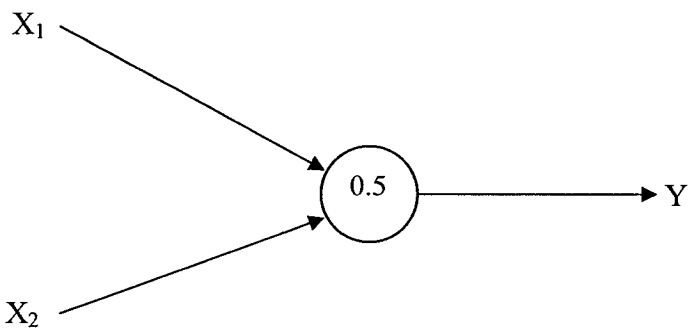


Figure 2.4 Perceptron attempting to express the XOR problem

This is above threshold, and so the output 1 is obtained. But the given perceptron fails for the last case which is when both inputs are 1, the output is 0. We now have a geometrical interpretation of the perceptron. A perceptron with weights w_1, w_2, ..., w_n and threshold T can be represented by a hyperplane. All points on one side of the

hyperplane belong to one class. The hyperplane (perceptron) divides the set of all points (patterns) into 2 classes. In the case of the XOR problem, there are 2 inputs. Therefore, we are able to have 2 dimensions. The points that we want to classify are [(0,0), (1,1)] in one class and [(0,1), (1,0)] in the other class. A hyperplane showing no hyperplane can divide the given set of points is shown diagrammatically in the figure 2.5 below. Clearly, we cannot classify the points (crosses on one side, circles on other) using a straight line. Hence, no perceptron exists which can solve the XOR problem.

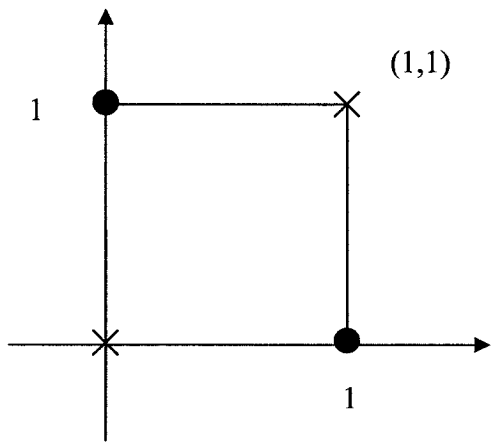


Figure 2.5 The XOR problem where no hyperplane can divide the given set of points into two classes.

Multi-Layer-Perceptron

The Multi-Layer-Perceptron was first introduced by M. Minsky and S. Paperta in 1969 [9]. It is an extended Perceptron and has one or more hidden neuron layers between its input and output layers. Due to its extended structure, a Multi-Layer-Perceptron is able to solve every logical operation, including the XOR problem. This perceptron is

composed of an input layer, a few hidden layers, and an output layer. They use the Delta learning rule which was developed by Widrow and Hoff [9]. The Delta rule is a supervised learning technique that is also referred to as the Least Mean Square (LMS) method, and is one of the most commonly used learning rules. For a given input set, the output set is compared to the correct answer. If the difference is zero, no learning takes place; otherwise, the weights are adjusted to reduce this difference. Multi-layer perceptron are of the feed forward type and utilize supervised learning techniques.

Hopfield Net

The Hopfield Net was first introduced by physicist J .J. Hopfield in 1982 [9] and belongs to the neural network types called 'thermodynamical models'. It consists of a set of neurons, which are interconnected. There is no differentiation between input and the output neurons. The main application of a Hopfield Net is the storage and recognition of patterns, e.g. image files. They are of the feedback type of networks and they use supervised learning rules.

Kohonen Feature Map

The Kohonen Feature Map was first introduced by Teuvo Kohonen [9]. It is probably the most useful neural net type when considering the simulation of the learning process of the human brain. The essence of this type is the *feature map*, which is a neuron layer where neurons organize themselves according to certain input values. The type of this neural net is both feed forward (input layer to feature map) and feedback (feature map). Furthermore, they use unsupervised learning algorithms.

2.2 Overview of Safety Critical Systems

Having discussed the ANN architecture, training process, and learning algorithms that were adapted, we now move on to the safety critical neural network domain and begin with introducing safety critical systems. There have been numerous techniques that have been discussed in literature about the safety related issues in a software system. These issues have discussed different aspects such as faults, hazards, etc. Discussing each of these techniques is beyond the scope of this thesis. We would just like to draw attention to some of the issues that have to be highlighted when we use ANN in safety critical systems.

2.2.1 Basic Definition

Computer, electronics, or electromechanical systems are those whose failure may cause injury or death to a human being. For example, an aircraft or nuclear power stations control system [11]. Thus, safety critical systems are systems where the system specification and design must be carefully executed in order to assure that no errors occur in the implementation.

Definitions of safety-related systems and components are as follows [11]:

• *Software system safety* –The software will execute within a system context without contributing to hazards.

• *Safety-critical software* – any software that can directly or indirectly contribute to the occurrence of a hazardous system state.

• *Safety-critical functions* – those system functions whose correct operation, incorrect operation (including correct operation at the wrong time), or lack of operation contributes

to a system hazard.

• *Safety-critical software functions* – those software functions that can indirectly, in consort with other system component behaviour or environmental conditions, contribute to the existence of a hazardous state.

2.2.2 Software System Safety Analysis.

Industries have devised their own form of safety standards [11, 12] for different applications. These safety lifecycles are applicable to systems solely involving hardware or software. The lifecycle must be managed through the stages of system development, acquisition, operation (includes maintenance), and disposal. The safety lifecycle's primary aim is to identify, mitigate and control hazards resulting from faults and failures. Effective execution of all required safety processes must be ensured by a plan devised by a safety management team.

Verification and Validation (V&V) plays a vital role in this place. Verification is a process of checking if the system conforms to the specification. Verification is to assure that the software components or software systems meet their specified requirements [13]. Specified requirements are not only found in requirements specification documents but can also be found in functional specifications, architecture and design models, test cases, and so on. The purpose of validation is to demonstrate that a software component or system fulfills its usage requirements when placed in its intended environment [13].

The process of V&V is applied to the last stages of the Software System Analysis. Some of the safety systems have quantitative requirements. Aircrafts for example have

such safety standards. Failure rates must occur with probability of less than 10^{-7} per hour and may be due to some technical fault in the overall system. This can be considered as a form of safety requirement which needs to be demonstrably shown to be achieved. Similar standards have also been expected out of space applications that use adaptive systems.

2.2.3 Challenges for Safety Critical Neural Networks

Artificial Neural Networks (ANN)-based systems have emerged as a powerful class of autonomous and adaptive systems [11, 12, 13]. Due to their learning abilities, ANN have been increasingly attracting attention in applications where autonomy is critical and where the identification of the possible fault scenarios is not exhaustive before hand. Though ANN has been accepted in many fields, there has been a certain concern as to their acceptability in safety critical systems. The application of ANN in safety critical system has to be supported by some techniques which can minimize the amount of risk that is involved. So, to meet the requirements of such systems, ANN have to be certified; however, there exist a number of issues that have currently not allowed ANN to be certified. Specifically, in this thesis, our objectives are to:

(1) Address issues in the V&V of the ANN.

(2) Discuss current approaches to the V&V of ANN and identify limitations of the conventional approaches to verification of ANN based systems.

(3) Propose an approach for V&V of ANN-based systems for on-board fault diagnosis in an autonomous system.

(4) Demonstration of the same with an ACS model and training the neural network to provide fault tolerance for the ACS model.

(5) Demonstrating rule extraction from trained neural network to enable comprehensibility of the neural network model developed.

(6) Demonstrating the use of formal verification to facilitate neural networks to be used in safety critical systems.

2.3 Conclusion

This chapter began with an overview of the ANN, their architecture, and topology. We then introduced the training and learning algorithms that are commonly used within the ANN. This led to the discussion of the feed forward type of training and back propagation algorithm with special emphasis, since this algorithm will be extensively used in our proposed approach which will be discussed in the subsequent chapters. We also discussed about safety critical applications and challenges for safety critical neural network and issues of such approaches. This discussion pointed out the importance of verification and validation of safety critical systems. The chapter concluded with a brief overview of previous approaches in the field or V&V of neural–network-based systems.

Chapter 3

V&V of Neural Networks: Overview and Proposed Approach

This first part of the chapter gives an overview of the concept of Rule Extraction, as well as, the algorithms available for Rule Extraction, which enables the extraction of rules from trained neural networks. The chapter further emphasises the need for development of ANN-based fault diagnosis system and the need for verification of such systems. The chapter then explains the proposed framework for verification of ANN-based fault diagnosis systems and formal verification of the same. We finally provide a brief description of UPPAAL and PVS; the tools used for correctness checking of the rules extracted using the rule extraction algorithm.

3.1 An Overview of Rule Extraction

ANNs have been widely accepted in industries due to their salient features such as knowledge acquisition, knowledge retention, and robustness. However, this success is at a cost of the ability of ANN to explain in a comprehensible manner the process by which the given decision has been taken. This explanation capability is an integral part of the functionality of ANN used in safety critical applications.

These explanation capabilities are defined by the rule extraction process. This can be defined as the process of deriving a symbolic description of a trained ANN [14]. Ideally, the rule extraction process results in a symbolic description which closely resembles the behaviour of the network in a brief and comprehensible form.

Several advantages result when using rule extraction in ANN, namely:

(1) User explanation capability: In order for adaptive systems, in particular ANN, to gain a higher degree of acceptance, they must incorporate some form of explanation capability. This should be a vital portion of the trained ANN. The explanation capability provides a means of understanding the symbolic and connectionist kind of approach used in Artificial Intelligence. Further, it opens up the black box module and explains the intermediate stages involved in the working of the ANN.

(2) Extension of ANN in safety critical domain: In addition to the user explanation capability, the need to verify the ANN architecture and topology is also an important facet of rule extraction. The requirements are for the ANN solution to be transparent so that the different states of the system are interpreted unambiguously. Such requirements will enable us to identify if ANN solutions are erroneous, as well as, why and when suboptimal solutions occur.

(3) Knowledge acquisition for symbolic Artificially Intelligent(AI) systems: Machine learning algorithms have overcome the problems of 'knowledge acquisition' which was believed to be the most time consuming and challenging task in building an expert system. Recent developments have shown that ANN can assist the task of knowledge acquisition, and outperform other symbolic machine learning algorithms.

(4) Software verification and debugging of ANN components in software systems: If the neural networks are part of a large software system which must be verified, then the neural network component must also be verified. In this case, the rule extraction

algorithm provides a mechanism for partial or complete 'decompiling' of the trained neural networks. This serves as an implicit way of achieving software verification.

3.2 Rule Extraction Algorithms

Till now, we have discussed the advantages of rule extraction from trained neural networks. We now present the different rule extraction algorithms present in literature for the same type of networks. There are a number of different techniques that have been developed to achieve the desired goals of extracting rules from trained neural networks. The rule extraction algorithms are classified by (a) the expressive power of the extracted rules; (b) the *`translucency'* of the view taken within the rule extraction technique of the underlying Artificial Neural Network units; (c) the extent to which the underlying ANN incorporates specialized training regimes; (d) the *`quality'* of the extracted rules; and (e) the algorithmic *`complexity'* of the rule extraction/rule refinement technique [15].

The two basic categories of rule extraction techniques *are `decompositional'* and *`pedagogical'*. There is also a third - labeled as *`eclectic'*, which combines elements of the two basic categories [16]. The decompositional approach focuses on extracting the rules at hidden and output units within a trained neural network. A basic requirement for rule extraction techniques in this category is that the computed output from each hidden and output unit in the trained neural network must be mapped into a binary outcome which corresponds to the notion of a rule consequent. All these individual binary rules extracted from the hidden and output layers are combined together to form a rule base. The particular interest in this type of classification are two techniques which are SUBSET algorithm and M of N technique, here the algorithm searches for a set of weight

containing the initial link/connection with sufficient positive value so that the threshold value on the unit which is being analyzed will be exceeded irrespective of the values in the other links or weights.

On the other hand, the pedagogical approach treats the ANN as a black box. The core idea in the pedagogical approach is to view rule extraction as a learning task, where the target concept is the function computed by the network and the input features. The core idea in the *`pedagogical'* approach is to view rule extraction as a learning task where the target concept is the function computed by the network and the input features are simply the network's input features' [5]. Hence the `pedagogical' techniques aim to extract rules that map inputs directly into outputs.

The third category in this classification scheme combines the elements of both the Pedagogical and Decompositional rule extraction techniques.

Rule extraction techniques can be classified into the following three categories. These will be explained in brief in the subsequent sections.

(1) Boolean rule extraction using decompositional approach

(2) Extraction of fuzzy rules.

(3) Boolean rule extraction using Pedagogical approach

3.2.1 Boolean Rule Extraction using Decompositional Approach.

The first sets of algorithms we discuss incorporate the decompositional approach. The backpropagation algorithm has been used successfully in problem domains involving learning and generalization. For this reason, there existed a need for developing a rule extraction technique for this algorithm which led to applying the decompositional

approach to such problems. The main focus was placed on extracting Boolean rules from trained neural networks. This is done at the individual (hidden and output) units [16] where the basic *motif* initially searches for sets of weights containing a single link/connection of sufficient (positive) value to guarantee that the bias on the unit being analyzed is exceeded irrespective of the values on the other links/connections. If a link is found which satisfies this criterion, it is written as a rule. The rules extracted at the individual unit level are then aggregated to form the composite rule base for the ANN as a whole. A schematic of the basic algorithms as reported by Towell and Shavlik [16] using Boolean extraction which in turn use decompositional techniques is been explained below.

SUBSET ALGORITHM:

For each hidden and output unit:

1. Extract up to S_P subsets of the positively-weighted incoming links for which the summed weight is greater than the bias on the unit. Here P is a set of positive weights and p is a element of the set S_P.

For each element p of the S_P subsets:

2. Search for a set S_N of a set of negative-weights, so that the summed weights of p plus the summed weights of N – n (where N is the set of all negative weights and n is an element of S_N) exceed the threshold on the unit;

3. With each element n of the S_N set, forms a rule: `if p and NOT n, then the concept designated by the unit'.

The subset algorithm has been successful in some cases; however, one of the major concerns with using these algorithms, as reported by Towell and Shavlik, is that

searching through the entire search space is time consuming. One option used in [17] for restricting the size of the solution search space is to place a ceiling on the number of antecedents per extracted rule [16, 17]. In such cases some of the rules may be omitted. Another important development in the decompositional technique is *M-of-N* Techniques [17]. The *M-of-N* concept expresses rules in the form:

If (M of the following N antecedents is true) then ...

The algorithm is explained in brief in the section below.

The M-of-N Technique.

1. Generate an Artificial Neural Network using the Knowledge based ANN system and train it using backpropagation. With each hidden and output unit, form groups of similarly-weighted links.
2. Set link weights of all group members to the average of the group;
3. Eliminate any group which does not significantly affect whether the unit will be active or inactive;
4. Holding all link weights constant, optimize biases of all hidden and output units using the back propagation algorithm;
5. Form a single rule for each hidden and output unit; the rule consists of a threshold given by the bias and weighted antecedents specified by the remaining links;
6. Where possible, simplify rules to eliminate superfluous weights and thresholds.

M-of-N technique has been tested successfully for varied range of problem domains including two from the field of molecular biology, such as splice junction problem and the promoter recognition problem [17].

3.2.2 Extraction of Fuzzy Rule

The extraction of Boolean rules using decompositional technique was growing within the scope of ANN. At the same time, the method of extracting fuzzy rules from trained neural networks was also making developments in the so called neuro-fuzzy systems. Neuro fuzzy systems have three distinct elements. The first is that it contains a mechanism to insert the knowledge in the form of fuzzy rules into an ANN structure .The second element is training the neural network, which in our case is tuning the membership function according to a pattern in the training data. The third element is extracting refined rules from trained neural network, where the knowledge is stored as membership functions. In order to continue making progress, substantial research is being conducted in this field.

3.2.3 Boolean Rule Extraction using Pedagogical Approach

In this implementation, the underlying Artificial Neural Network is treated as a *'black box'* which contains a set of rules from a medical diagnostic problem domain, which are extracted from changes in the levels of the input and output units. One of the earliest published pedagogical approaches to rule extraction was the Validity Interval–Analysis (VIA) technique developed by Thrun [18]. This approach extracts rules that map the input and output directly. This algorithm uses *generate* and *test* procedures for the extraction of symbolic rules from a trained neural network which was trained using standard back propagation approaches. This algorithm differs from other techniques wherein the input checks for the individual activation of the units. In this technique, the emphasis is on what is called the 'validity intervals'. Here the validity interval of a unit

specifies the maximum range for its activation value. The only assumption that is made about the network is that the non linear transfer function of the units in the network are either monotonic or continuous, which is the case in standard backpropagation algorithm. The algorithm adapted for rule extraction is described below:

VIA Algorithm

1) User assigns arbitrary intervals to all units in the neural network.
2) VI-Analysis is done on these units and the intervals are being refined.
3) Refining the intervals is done by iteratively detecting and excluding the activation functions that are inconsistent with the weights and biases.
4) The activation functions that are consistent with the initial input interval will also be consistent with the refined interval.
5) The possible outcome of the VI analysis would be either consistent with the input or it would return an empty interval.

This algorithm is explained through an example. Activation values are denoted by x_i where i refer to the index of the unit in the network in equation 3.2. If unit i is an input unit, its activation value will simply be the external input value. If not, i refer to a hidden or an output unit; Let $P(i)$ denote the set of units that are connected to unit i. The activation x_i is computed in two steps:

$$net_i = \sum_{k \in P(i)} W_{ik} x_i + \theta_i \qquad (3.1)$$

$$x_i = \sigma_i(net_i) \qquad (3.2)$$

Where the net_i is called the net-input as shown in equation 3.1 and W_{ik} and θ_i

represents the weight and the bias respectively, σ_i denotes the transfer function which was inferred from the training of the backpropagation algorithm, that is shown in equation 3.3

$$\sigma_i(net_i) = \frac{1}{1+e^{(-net_i)}} \quad \text{with} \quad \sigma_i^{-1}(x_i) = -\ln(\frac{1}{x_i} - 1) \qquad (3.3)$$

The validity intervals for the activation function x_i are denoted by $[a_i, bi]$. There are two phases in the VIA algorithm: forward phase and the backward phase. Usually the network topology includes an input layer and output layer connected through a hidden layer. Furthermore, there is a single layer of weights connecting the two layers. Refining the validity intervals is done by means of a technique called *linear programming*, such as the *simplex algorithm.* All of the non-linearities in the transfer function must be removed, and thus the refinement of the intervals is done by projecting back the net-interval. The resulting validity intervals form a set of linear constraints on the activation values. All these if-then rules are then written of the form of a target class C, and where $I = [a_i, b_i]^m$. Here, the precondition-hypercube of the rule to be verified is given by I and

If input ϵ some hypercube I, then the rules belong to a class C

If input ϵ some hypercube I, then the rules do not belong to a class C

Thus the VIA algorithm allows us to check for the correctness of the extracted rules by verifying the correctness of the resulting classes.

3.3 Need to Develop Fault Diagnosis Systems

Safety critical systems are currently being employed in many fields, such as

medicine and defense. These systems are vital, and thus failures in such systems may lead to fatal consequences. Therefore, one attempts to use certain techniques which can detect faults in advance and tolerate them before they develop into failures.

Software fault detection or diagnosis is termed as the use of any technique or procedure that aims at detecting errors during any phase of a software life cycle thereby avoiding any faults in safety related applications [19]. The traditional development approaches aims at:

- Avoiding the introduction of faults in the system
- Removing faults during subsequent verification

Fault tolerant approaches can suppress the effects of such faults in the system [20, 21, and 22]. Typically they are based on the concept of code redundancy, which is making sure that there is a redundant backup approach available in case the present technique fails. The main objective of such fault diagnosis systems is that software faults should not give rise to system failures. The faults must be detected before it starts affecting a certain part of the system such that the result produced conforms to the intended function.

Safety critical systems are influenced by several external factors as shown in figure 3.1. These factors are responsible for affecting the safety of such systems [13] and have incorporated different standards depending on each industry. Faults can occur at any part of the software life cycle and removing such faults and maintaining the safety of such systems is a huge challenge.

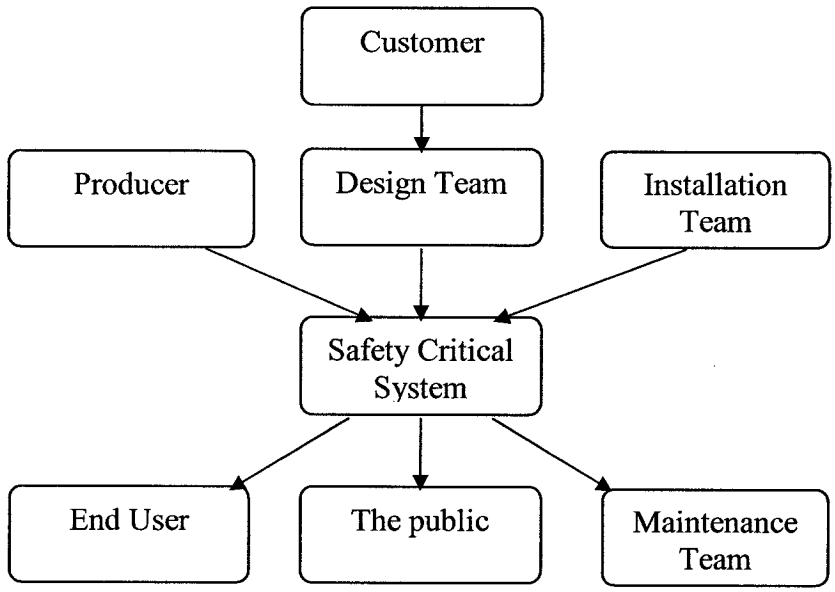


Figure 3.1 Different bodies that influence safety critical systems [13].

So it has been established that various factors affect the safety of such mission critical applications and various faults can occur at different stages of the software lifecycle. Thus, a well-tested framework to diagnose, avoid, and tolerate faults is required. Safety evidence plays a major role in arguing that the system has met all safety requirements. At every phase of the software lifecycle such evidence has been extracted and thus the cost of safety is inevitably expensive.

3.3.1 Development of ANN-based Fault Diagnosis Systems

Once the concept of ANN and safety critical applications has been understood, we can relate both of them and understand the role of ANN in safety critical systems. The adaptive behavior of ANN has attracted much attention from researchers however the role of ANN in safety critical applications to date has been very limited. We introduce

this aspect through examples.

Knowledge based artificial intelligent systems are popularly being used in safety critical applications. The field of medicine has a keen interest in both neural networks and knowledge based expert systems for diagnosing and advising doctors. The product created for this diagnosis is called PRO*forma* [23, 24] and has many benefits such as:

- Representation and accessibility of ever-increasing knowledge and information
- Dealing with altered or updated information
- Improved time resources, which are required to perform certain functions

This product learns from examples and feeds the new set of knowledge back into the system. The diagrammatic representation of the *PROforma* domain model is given in figure 3.2. The components of the system include **Beliefs**: Facts known by the system or fed into the system. **Goal**: The intended goal of the system. **Options**: The possible options available in order to achieve the goal. **Argument**: Argues how the chosen option achieves the goal and why it is more beneficial than others. **Processes**: The resultant processes that have been derived. **Actions**: The actions performed on the processes, with resultant facts fed back into the system.

The common problem with this structure arises at the 'actions' stage. This stage restricts the role of the system to 'advisory' roles, where the medical consultant ultimately has the final decision. Thus, the advisory role of a knowledge-based artificial intelligence system is very limited. Consultants may become too dependent on the system and compromise final decisions [24]. This problem exists across a range of other applications. Another problem is the generalization, which is the ability to give a correct output when provided with unseen or novel inputs.

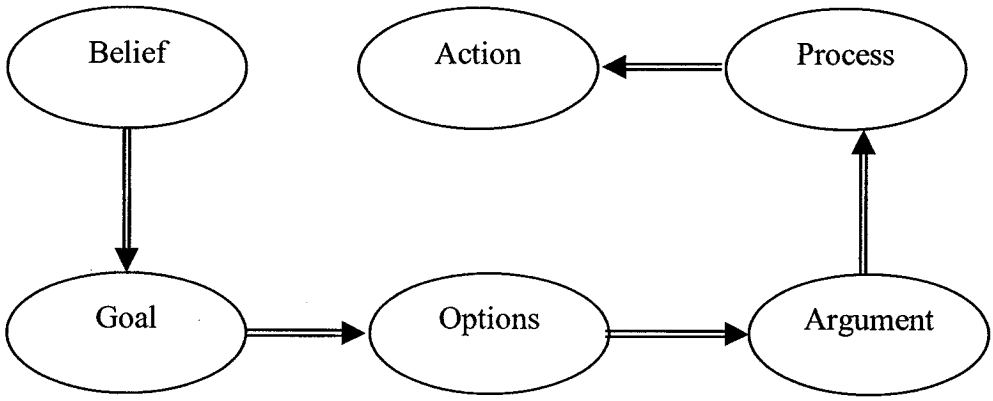


Figure 3.2 Domino model used in *PROforma* [22]

A more specific example related to neural networks is concerned with Statistical Process Control (SPC) [25]. The end-application in the case of an SPC may be a production plant with ANNs monitoring process operation and performance. The major advantages of this type of function includes that it is capable of detecting the occurrence of unspecified or undesired production. Important process disturbances and process malfunctions (or faults) can also be detected [25]. This can lead to great improvements in the overall system process operation. Siemens has been working on producing intelligent fire detectors [25]. These detectors use both neural networks as well as data that have been gathered from fire tests over the past several years. ANNs are used to combine measurements of smoke density and temperatures from optical and heat sensors. ANNs are then trained using a collection of rules discovered from human experts over many decades. This product is called 'Algorex' and the developers are so confident in its ability that they are offering compensation for any damages resulting from it. From these examples, it can be seen that there is a need for ANN-based systems to be dependable and safe in order for them to be used in safety critical applications.

3.3.2 Need for Verification of ANN-Based Fault Diagnosis System

The systems that have used neural networks to date have been using them only in advisory roles where they have limited flexibility. The concern behind this is that there is a large amount of uncertainty with respect to the adaptability of ANN. This further leads to the reason why neural have not been fully accepted within the study of safely critical system applications. The issues related to the adaptability are the following:

- Correct functioning of the system according to its design
- Providing that the design meets its requirements
- Fault tolerance

Within the context of flight control systems, neural networks have high dependency roles. Thus, the desired aim for ANNs is to achieve both critical and essential roles.

- *Critical* – a function in which the occurrence of a failure would prevent proper operation of the system [13].
- *Essential* – a function in which the occurrence of a failure would reduce the ability of the safety-critical system to handle operating conditions outside of the normal operating environment [13].

Given the lack of certification methods, developers of safety-critical systems are required to build AI components that are recommended to perform the following [13]:

1. Redesign the proposed system without the AI component.
2. If the above is not possible, isolate the AI technology from non-AI components based on hazard and system safety analysis and other processes.
3. If the above is not possible, then withdraw from the project quoting 'Codes of

Practice'.

The problem of verifying that the network is actually a refinement of the specification needs to be tackled [13]. This may require formal specification or otherwise to define the behavioral requirements of the network [13]. Exhaustive testing may be impractical if the input space is too large and complex, since ANNs cannot be easily divided into small meaningful components [13] which lead to the black-box representation of systems.

3.4 Proposed Approach

After analysis of the use of neural networks in safety critical systems, it is evident that there is a need for a complete framework for the verification of neural network-based software systems. Formal methods could be used successfully for verifying neural network-based software systems. Neural networks have been applied successfully in many real world scenarios, however when the systems become increasingly complex and when the input/output mapping becomes difficult then it is difficult to understand what the network has learned. Our framework is, thus, an attempt to understand the black box. The block diagram of the proposed framework is shown below in figure 3.3.

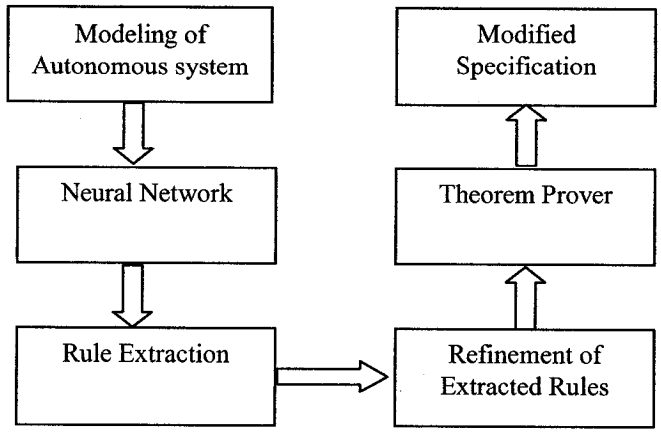


Figure 3.3: Block diagram of the proposed approach

Each of the modules of the proposed approach is explained in detail subsequently. The Modelling of the Autonomous system under consideration for our framework is the Attitude Control Subsystem (ACS) Model in simulink. The usage of the Altitude Control Subsystem (ACS) [26] is that it is commonly considered as momentum management system, which orients the main structure of the satellite at desired angle(s) within the required accuracy. This 'required accuracy' is set by the payload, communication devices, etc, mounted on the main structure. The attitude of a spacecraft may be specified in a number of ways such as direction cosines, Euler's angles, etc. The bodies in space are constantly subjected to disturbance torques by various sources. In order to combat those disturbances, an ACS subsystem is required. The torques usually re-orients the spacecraft unless they are restricted in some way or another. Hence, it is essential that the spacecraft determine its attitude using sensors and controls them using actuators.

The major components of the ACS are [26]: the Attitude Control Processor (ACP) or on-board attitude controller, control torquers or actuators, (for example, reaction wheels (RW), momentum wheels (MW), magnetic torque bars (MTB), etc.), attitude

sensors, and the spacecraft body. The attitude sensors acquire the spacecraft's altitude. The errors in the angles are computed and based on these error signals the on-board ACP generates torque command voltages. Control actuators produce torques depending on the torque demand/command voltage inputs they receive from the ACP. In this process, the required attitude is attained. A generic ACS block diagram with the reaction wheel as the actuator for control along a single axis is shown in Figure 3.4:

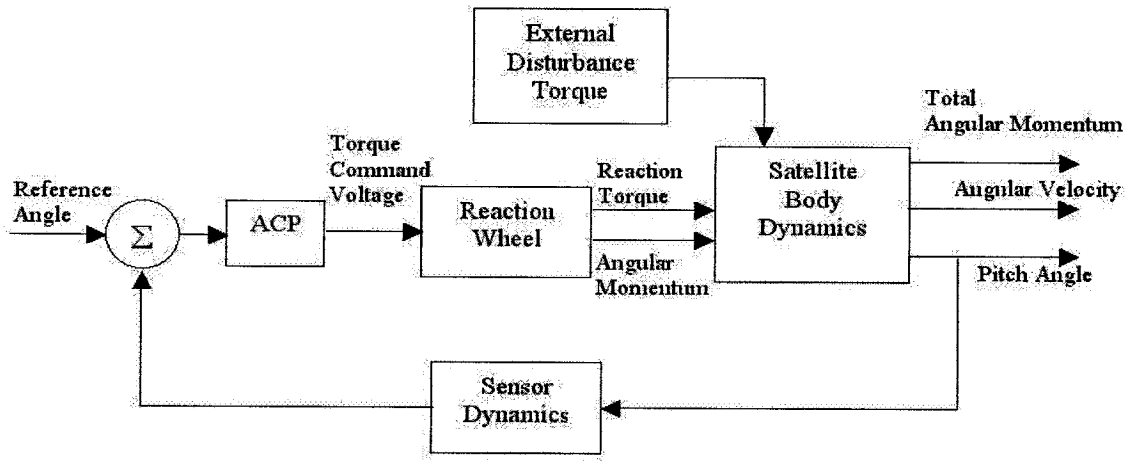


Figure 3.4 ACS block diagram [26]

A MATLAB-*Simulink* model of a generic ACS of a satellite has been developed. *Simulink* was chosen as the modeling tool because of its wide acceptance and growing demand in system modeling, analysis and design.

The model has been developed for fault-diagnosis in the ACS along a single axis, i.e., along the pitch axis. Thus, it is not necessary to consider any *cross-coupling* effect. The actuator or reaction wheel block in the control loop of the developed ACS model is primarily based on the reaction wheel model presented in [14]. The model has been extended and modified in order to include fault injection capabilities which will be

presented in subsequent discussions. Also, an ideal dynamics for altitude sensors has been assumed, i.e., signals from sun sensors, horizon scanners, magnetometers, etc. have been fed back to ACP without any error or time delay. Therefore, the gain of the *Sensor Dynamics* block has been assumed to be 1. Data was generated from the model under three different failure scenarios both with and without the injection of faults. The symbolic information that is gathered from the ACS model is then fed into the neural network and the trained neural network is applied to rule extraction algorithm.

Having generated the required data from the model, the next step was training the neural networks using these data. The trained neural network is then applied to a rule extraction algorithm. We had started with a decompositional approach and chose the SUBSET algorithm [17] to extract rules from the trained neural network. A simple SUBSET breath first algorithm starts with searching to see if there are any sets containing a single link which can indeed guarantee that they exceed the threshold. If so, then this set is written as a rule. The search proceeds by increasing the size of the subset until all the possible subsets have been completely explored. Once these rules have been extracted the subsumed and overly general rules are removed and finally the rules are derived. Though the SUBSET algorithm worked quite well with binary data, the number of rules extracted could be numerous. On comparing the input and the target after rule extraction, the amount of error that we had encountered was slightly high. The rules that were extracted were further checked for correctness using the model checker, such as UPPAAL.

The other rule extraction algorithm used is based on Pedagogigal approach, where the knowledge may be in terms of symbolic information (such as *if-then* rules). The

decompositional approach is of particular interest to safety critical applications since it combines the symbolic knowledge and neural network learning paradigm. By combining these two features we can refine and modify the symbolic knowledge. The block diagram showing the process of combining the symbolic knowledge and neural network learning is shown in figure 3.6.

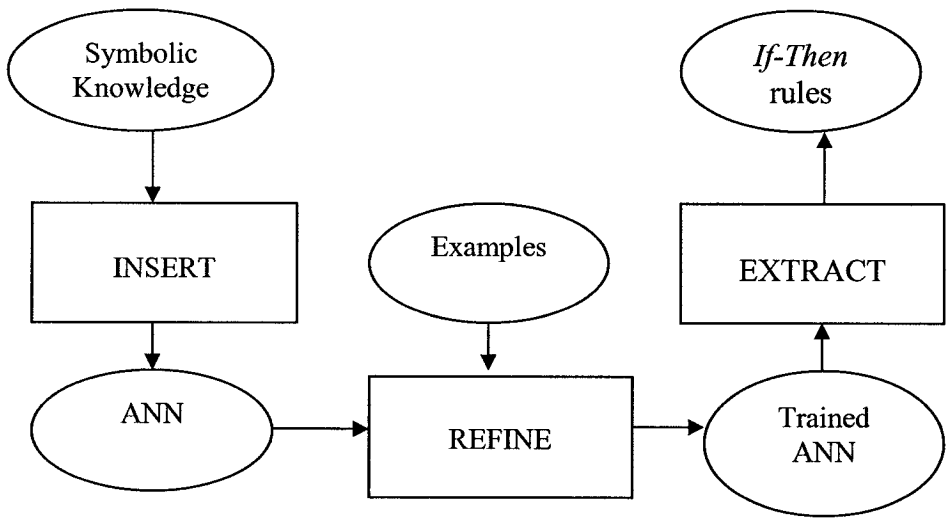


Figure 3.5 Framework of combining the symbolic knowledge and neural network learning [27].

The term referring to the combination of symbolic knowledge with neural learning is 'hybrid'. There are three main stages associated with the framework for 'hybrid' ANNs. The first stage involves gathering initial symbolic information. This initial information is prior knowledge known about the problem involved and can be in the form of rules. The initial rule set is then 'inserted' into a neural network structure. This can be achieved through rule-to-network algorithms [27, 28], which map parts of

each rule into the internal structure of the ANN (such as weights and neurons).

Once the ANN has been represented as rules or in other words once the symbolic information has been inserted, the next stage involves refinement or learning. Refinement uses training samples to add or modify new knowledge to the initial symbolic information. This is the key motivation for adopting the neural learning concept. Training data may be simulated or may consist of data sampling gathered from real-world interaction. A typical algorithm used for the learning process is the back-propagation algorithm. During refinement or learning, the topological structure of the ANN may require adaptation to accommodate changes in the initial rule set. For safety-critical applications, 'hybrid' ANNs encompasses the potential to overcome many problems associated with typical ANNs.

Analytical arguments about the solution provided by the 'hybrid' ANN may be based upon symbolic information. This enables a higher level of abstraction as opposed to dealing with low-level representation, such as weights, links, and neurons. For example, rather than using an overall output error (produced over some test set), the network can be analyzed in terms of the rules it embodies. Consequently, this provides extra comprehensibility for analytical processes.

Once the learning phase is complete, the final stage involves extracting symbolic knowledge from the final ANN. This process attempts to determine all changes or additions made to the initial symbolic information. In our approach, we use the VIA algorithm as a means of extracting the symbolic knowledge stored in the network. Once the final symbolic information has been determined, these rules which are of the *If -then* form are then mapped into a form understandable to the theorem prover, such as

Prototype Verification System (which is described in detail in section 3.4.3). PVS checks for the correctness and completeness of the extracted rules [30]. The output from the theorem prover helps us modify the specification of the model.

3.5 Formal Verification

Formal methods in software development are defined as "A method that provides a formal language for describing a software artifact (e.g. specifications, designs, source code) such that formal proofs are possible, in principle, about properties of the artifact so expressed" [29]. Formal methods involve the essential use of a formal language. These methods of reasoning are exemplified by formal proofs. A proof begins with a set of axioms, which are to be taken as statements postulated to be true. A set of inference rules must be given in each formal method. A proof then consists of a sequence of well-defined formulae in which each one is either an axiom or derivable by an inference rule from previous formulae in the sequence. The last formula in the sequence is said to be proven.

Formal methods in neural network represent the knowledge in specification languages and methods of representation such as rule extraction and decision trees. In our approach, we have chosen to use rule extraction to extract the symbolic knowledge from the trained neural network after the fixed neural networks are tested and refined by inserting examples. The testing and refinement provides us with test cases and test procedures. The symbolic knowledge that we have extracted from the trained neural network mimics the functionality of the network, these rules can be compared with the original requirements. This is done by representing them in a formal language in the form

of theorems and they are verified using a formal verification tool, such as PVS. These formal methods allow the system to be proven to be correct, complete, and consistent. There has been limited work done in the field of formal verification of neural networks and the formal methods that are available for the V&V of neural networks are constrained for specific networks only. Thus, our approach provides both the flexibility of usage on any type of network along with a defined level of formalization with respect to the rule refinement.

3.6 UPPAAL Overview

UPPAAL is a tool box [29] for modeling, simulation, and verification of real-time systems, based on constraint solving and on the fly techniques developed by Uppsala University and Aalborg University. It is suitable for those systems that can be modeled as collection of non-deterministic processes with finite control structure and real valued clocks, communicating through channels and/or shared variables. UPPAAL has three main parts in the form of a description language, a simulator, and a model-checker. The description language is a non-deterministic guarded command language with data types. It serves as a modeling or design language to describe system behavior as networks of timed automata extended with data variables. System behavior can be analyzed interactively and in automated fashion by manipulating and solving constraints that represent the state space of a system description.

3.6.1 Modeling in UPPAAL

The model consists of a collection of timed automata extended with integer

variables in addition to clock variables. The edges of the automata are represented with three label types: guards, synchronization action, and invariants.

Guards: Guards are conditions on the values of clocks and integer variables that must be satisfied in order for the edge to be taken. They are basically conjunctions of time and data constraints, where a timing constraint is of the form $x \sim n$ or $x - y \sim n$ (where n is a natural number and ~ represents <=, >=, =, <, >) and a data constraint is of a similar form $i \sim j$ or $i - j \sim k$ but with k being an arbitrary integer.

Synchronization: Automata communicate through communication channels. If 'a' is the communication channel between two processes then synchronization actions are represented by a! (Sending on channel a) and a? (Receiving on channel a).

Invariants: Control nodes are decorated with invariants to enforce progress in a system, which are constraints on the clock values in order for control to remain in a particular node.

3.6.2 Simulation and Model-checking

The simulator allows the user to examine the dynamic behavior of the system in an interactive and graphical manner. It explores only a particular execution trace, i.e. a sequence of states of the system. This provides an inexpensive means of fault detection in early stages of the design. Its is also used to visualize a diagnostic trace generated by the model checker so that the user can in an interactive and graphical fashion examines the execution trace which may result in a system error.

The model checker covers the exhaustive dynamic behavior of the system by

exploring the whole reachable state space of the system. It allows checking for invariant and reachability properties; in particular, whether certain combinations of control-nodes and constraints on clocks and integer variables are reachable from an initial configuration. Other properties, such as liveness properties, can be checked by reasoning the system in the context of testing automata or simply decorating the system description with debugging information and then checking reachability properties.

3.7 PVS Overview

The tool used within our proposed approach is Prototype Verification System (PVS), which provides a mechanized support for formal specification and verification. It is built by SRI (Stanford Research Labs) and uses tools to support formal methods. PVS is mainly intended for the formalization of requirements and design-level specifications, and for the analysis of intricate and difficult problems. PVS and its predecessors have been primarily applied to algorithms and architectures for fault-tolerant flight control systems and to problems in hardware and real-time system design [30].

PVS consists of a specification language, a number of predefined theories, a theorem prover, and various other utilities. The specification language used in PVS is based on classical, typed, higher order logic. PVS specifications are organized into parameterized theories that may contain assumptions, definitions, axioms, and theorems. The goal of the PVS theorem prover is to support efficient software development of readable proofs of all stages of proof development lifecycle. Thus, PVS is used as a theorem prover in our proposed approach to prove for the correctness and completeness of the symbolic rules that were extracted from the neural network.

3.8 Conclusion

This chapter provided an overview of the rule extraction algorithm and discussed the fault diagnosis aspect of the neural network. Furthermore, we discussed the proposed approach for verification and validation of neural network-based software systems. Finally, an overview of verification tools, such as UPPAAL and PVS (the tools used within our approach for both formal specification and verification of our system) was given.

Chapter 4

Verification of the ACS Model

The first part of the chapter gives an overview of the ACS failure scenarios under which the data was made available for training. This overview/discussion is based on the work reported in [32]. We then move on to the specific discussion of the rule extraction algorithm adapted in our proposed approach. We then explain in detail the aspects of verification introduced during training and rule extraction from the trained neural network. The specifics pertaining to the different formal techniques that were applied to formally verify the rules extracted from a trained neural network have also been shown in detail.

4.1 ACS Failure Scenarios

The training data which resulted for training the neural network was taken from the ACS Simulink model developed in [26]. The Simulated data was given as input to the neural networks, and the neural network output is compared with the simulated ACS model output. Thus Failure Scenarios were created by injecting faults in the ACS Simulink model. We wanted to train and extract rules from the neural network under different scenarios of failure. Three failure scenarios for which data have been provided for training are presented in the subsequent sections.

4.1.1 ACS Failure Scenarios -1

A random increase in the reaction wheel motor current

Hardware level failure in the motor driver unit (MDU) of the reaction wheel can cause this type of faults. This fault is injected when the Reaction Wheel is running near zero speed. The purpose of this fault is to represent failure under a surge in the current. The system behavior (pitch angle error) during the fault-free condition as well as under the presence of this fault can be observed in figure 4.1.

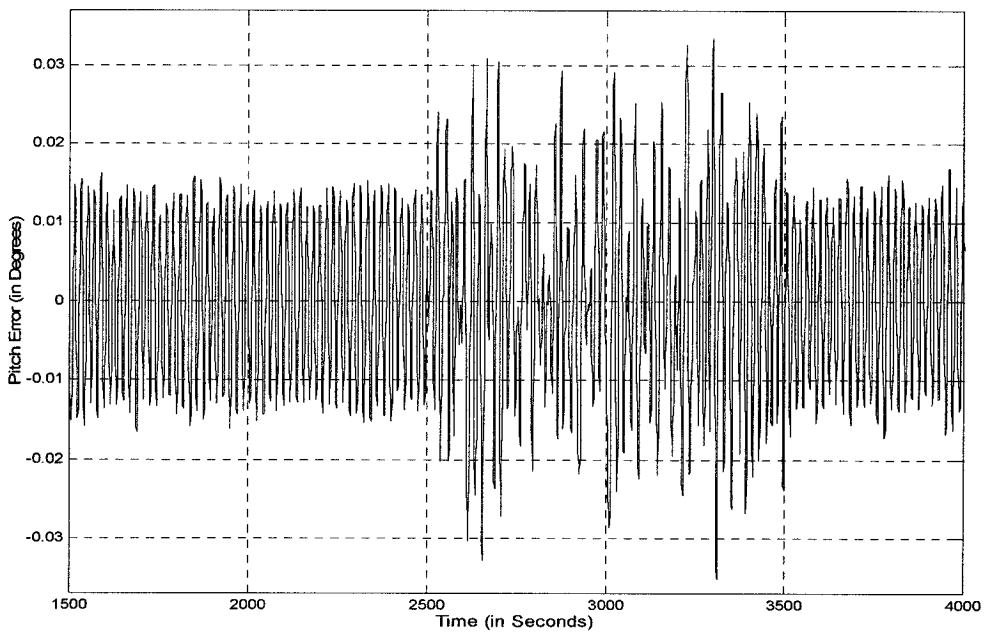


Figure 4.1: Pitch error vs time under failure scenario-1 [26]

The fault has been injected between *t=2500* and *t=3500* seconds. For all other intervals outside of this time range, the system behaved normally.

4.1.2 ACS Failure Scenarios -2

An Increase in Friction in the reaction wheel

These faults were injected when the Reaction Wheel was running near zero speed. The purpose of this fault is to represent a failure if the friction is increased in the wheel bearings because either the bearing material is wearing out or there is some problem with the lubricant flow. The system behavior (pitch angle error) during fault-free condition as well as under the presence of this fault can be observed in Figure 4.2.

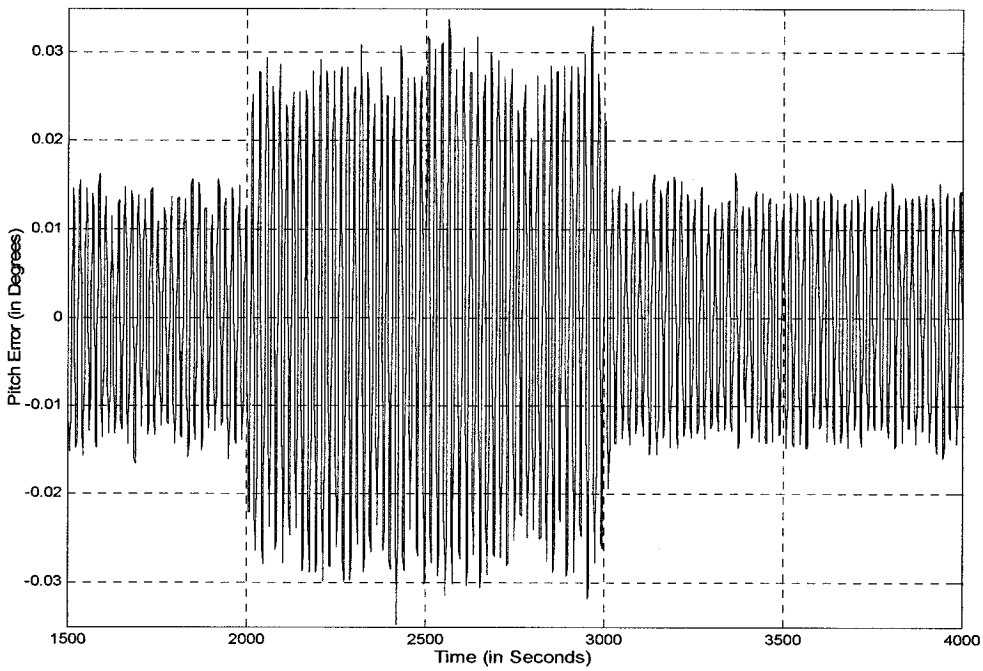


Figure 4.2: Pitch error vs time under failure scenario-2[26]

In this case, the faults were fed when t=2000 and it removed when t=3000. At all other stages, the system reacted normally.

4.1.3 ACS Failure Scenarios- 3

Bus Voltage Failure at High Speed

This fault has been injected when the Reaction Wheel was running near a maximum allowable speed. This type of fault may occur at low bus conditions when large back-EMF, developed in the reaction wheel motor operating at a high speed, limits the motor current, consequently the motor torque. The system behavior (pitch angle error) both under fault-free condition as well as under the presence of this fault can be observed in figure 4.3. The fault has been injected between t=2000 and t=3500 seconds. As anticipated, the system behaved normally outside this time range.

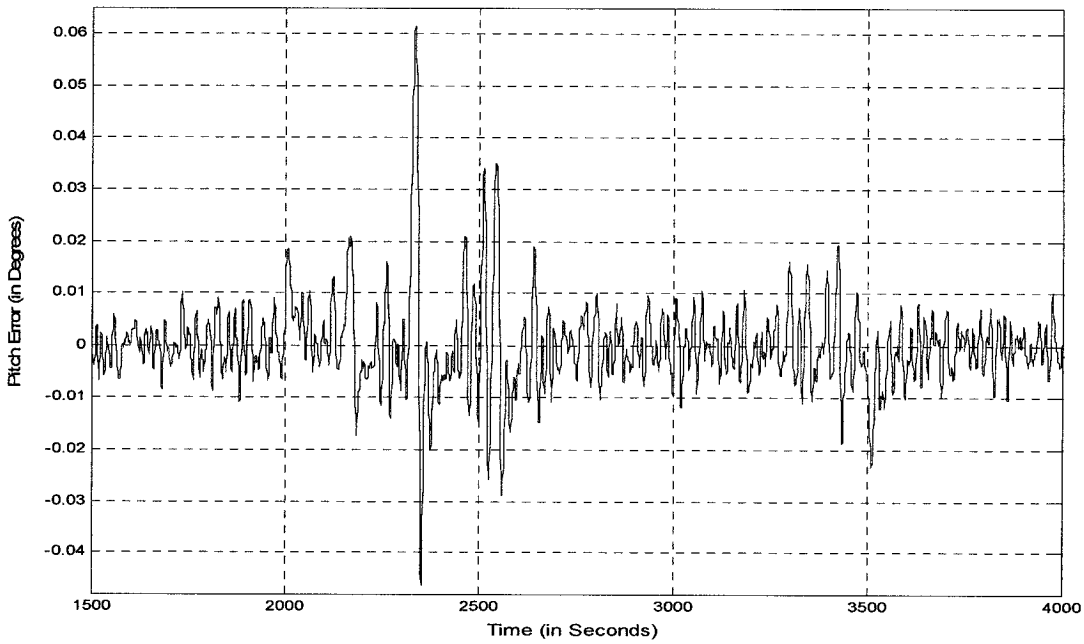


Figure 4.3: Pitch error vs time under failure scenario-3[26]

It should be clear at this stage that failure due to bus voltage may take place only at high operational speed of the reaction wheel. At low or near zero speed, even if the bus voltage drops to a value as low as 10 volts, the torque may not be limited because of

small back-EMF developed in the motor. And it is very unlikely that the bus voltage will ever reach such low values. For this reason, we should only consider scenarios at high speed. It has been assumed that the bus voltage level is 15 –28 volts under normal system conditions. Consequently, it has also been assumed that the maximum allowable reaction wheel speed is 5095 RPM which can be rounded to 5100 RPM. Figure 4.4 shows different ACS parameters and their behaviour under failure scenario 3.

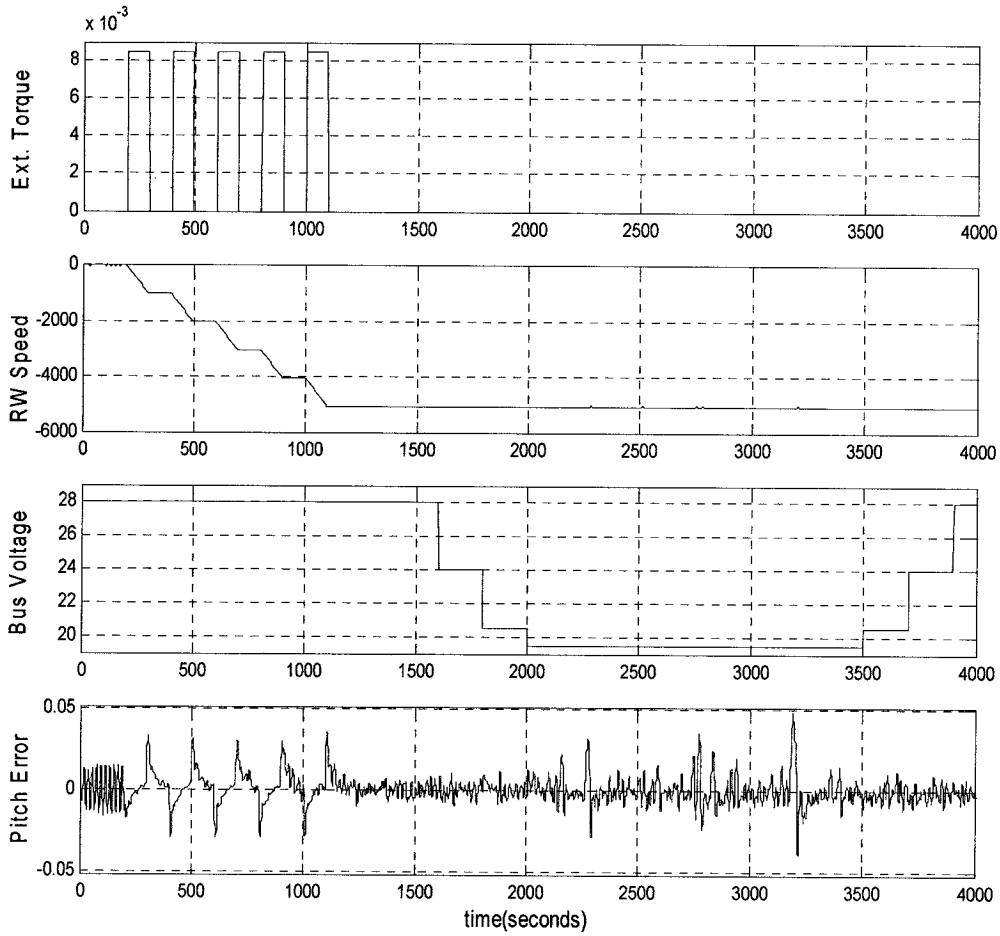


Figure 4.4 Different ACS parameters under failure scenarios 3[26]

4.2 Neural Network for the ACS Simulink Model

As discussed in Chapter 3, a multilayer perceptron model is generated for the Attitude Control Subsystem of a satellite which uses the backpropagation learning algorithm in supervised mode. A multilayer perceptron is a feed-forward neural network consisting of a number of neurons typically arranged in layers namely the input layer, hidden layer and output layer. The input layer receives the input and passes it on to the next layer using the weighted connections of the hidden layer. The structure of the neural network model that we had generated has a structure such as the shown in figure 4.5.

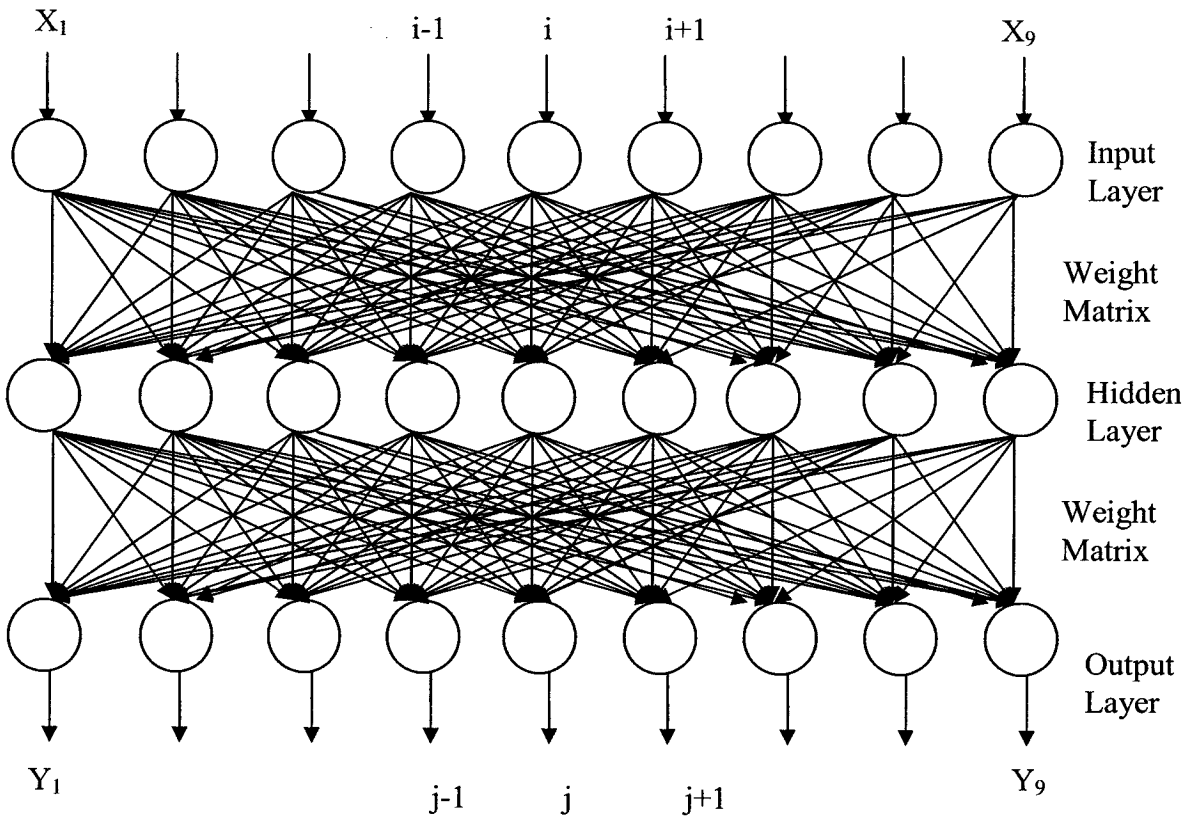


Figure 4.5 A multilayer perceptron architecture for the model chosen for training

In the neural network model structure shown in figure 4.5, each neuron takes one

input variable ranging from X_1, X_2.....X_9. These inputs represent Bus voltage, External Torque, Pitch, Pitch Error, Wheel Speed, Vehicle Angular Velocity, Motor Current, Motor Torque, Torque command voltage. The outputs are Y_1, Y_2....Y_9 which would be the trained output which tolerates the faults which are introduced in the input due to the three faulty scenarios considered.

Each of these nodes in the neural network has variable weights and a predefined transfer function between them. The input that is given to the neural network is propagated through the different layers until it reaches the final layer which generates the output of the neural network. The network output is then compared with the expected output and the mean square error is calculated. The error is then propagated back through those network layers which have contributed to the output. The process is repeated until a reasonable error is achieved. Thus the neural network which was given an arbitrary input resulted in the correct output after training. The training is done using the learning rule and the transfer function. The learning rule that was used aided in changing the weights and bias values. The transfer function is the function that relates the neuron output to the net output. The neuron output is calculated using the formula shown in equation 4.1.

$$x = \sum_{i=0}^{N-1} (W_i x_i - \theta_i) \qquad (4.1)$$

where W_i is the weight in the ith neuron on the neural network, xi is the input at the ith neuron, and θ_i is the bias value at that same neuron.It is assumed that i denotes neurons in the input layer of the neural network architecture shown in figure 4.5 and j denotes neurons in the hidden layer.

The transfer function or activation function that was chosen for the network is the sigmoidal function. Sigmoidal function is a mathematical function that a neuron uses to

produce an output referring to its input value as shown in equation 4.2.

$$Y=transf(x)\text{-}transf(\sum_{i=0}^{N-1} W_i x_i - \theta_i) \quad (4.2)$$

The activation function that is chosen is either tanh, logistic or Gaussian. In our case the bipolar *tanh* has been chosen as the activation function of the hidden units. The *tanh* produces both positive and negative values which results in faster training. The learning rate controls the rate at which the error modifies the weights.

In our training, deciding when the training is optimal and deciding on a stopping criteria were the two main obstacles. These were overcome by using the technique of early training and stopping as explained below. The work done by Vapnik [31] had shown that cross validation with the testing data is mandatory due to issues relating to overtraining and cases where the network starts to memorize the input data.

Figure 4.6 illustrates that the error decreases over a number of iterations (training each data sample) but then starts to increase after a point *x*. This problem can be solved by stopping the training at point *x* or when the smallest error has been detected (using the validation set as a reference).

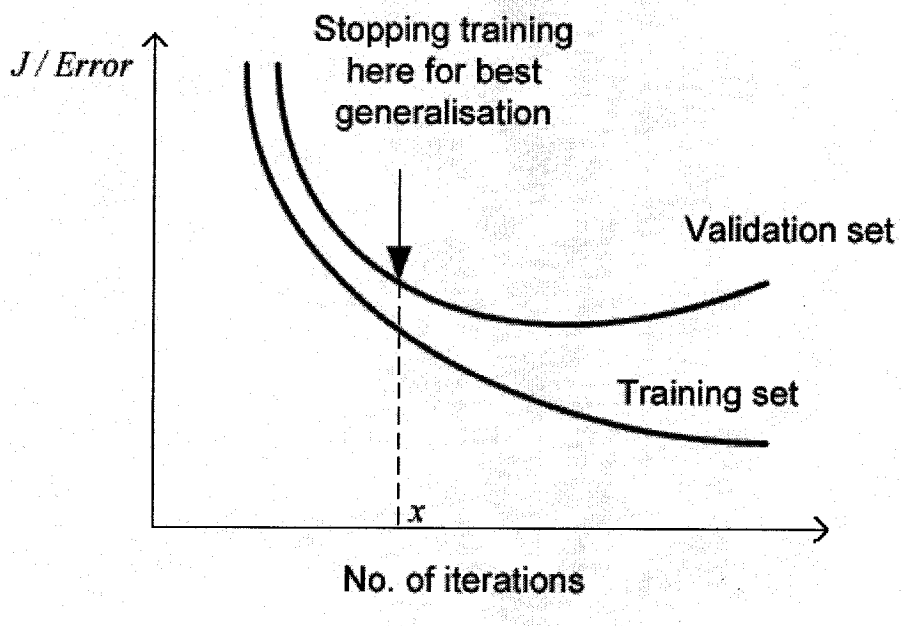


Figure 4.6 - Performing early training stopping to ensure best generalizations [31]

The following steps were followed to achieve maximum generalizations [31, 32]:

1. We divided the training data into *training* and *validation* set. The recommended size of the validation set was 10% of the total training data samples.

2. After a few iterations of training, the network error was measured using the validation set. Maximum error was considered for safety-critical systems rather than average error functions.

3. We stopped training when the error started to increase.

The neural network was developed using a neural network toolbox however prior to that we discuss the pre processing and the post processing that was required before the data generated from the ACS subsystem of the satellite was given to the neural network model.

4.2.1 Preprocessing and Post processing

The neural network that was used utilized was a feed forward type of backpropagation algorithm. For the architecture that was chosen, the data was in decimal format and we converted it to binary input so that it was comprehensible to the neural network.

The neural networks' training can be made more efficient by performing preprocessing steps on the networks input and target values. The data available to us from the ACS simulink model had nine variables out of which faults were induced into the three variables discussed earlier. The threshold values of the data variable considered as the input to the ANN are given in the Table 4.1 below.

Variables	Range		Unit	Comments
	From	To		
Bus voltage	+15	+28	V	Assumed range
External torque	$-10X10^{-3}$	$+10X10^{-3}$	N-m	Assumed range
Motor current	-0.95	+0.95	A	None
Motor torque	$-27.55X10^{-3}$	$+27.55X10^{-3}$	N-m	None
Wheel speed	-5200	+5200	RPM	None
Torque command Voltage	-5	+5	V	None
Pitch	-2	+2	Degrees	Approx range
Pitch error	-0.15	+0.15	Degrees	Approx range
Vehicle angular velocity	-0.0021	+0.0021	Deg/sec	Approx range

Table 4.1.Threshold values for data variables considered as the inputs to the ANN.

Since the network only accepts binary data, the decimal numbers were first converted into the accepted binary form. The values of some of the input variables were

real numbers and so these were converted to decimal first and then to the respective binary numbers in order to meet the requirements of the neural network at hand. The one main drawback in this type of training is that negative numbers could not be expressed easily. For experimental purposes we considered them as positive and then negated them back after post processing. The post processing process was the reverse process in which the binary data was converted back into the equivalent decimal numbers.

4.2.2 Neural Network Training using the Simulation Results

After completion of the pre processing stage, the ANN was trained for the first failure scenario where faults were induced in the motor current. The data of the motor current is considered individually for training. The preprocessed data are fed in as the input and target of the neural network and the neural architecture mentioned previously was established. The data variables are then trained under different training parameters to get optimum training. The parameters that were chosen for optimum training are shown in detail in Table 4.2 below.

Input	**10**
Hidden	**1**
Output	**10**
Learning coefficient of input	**0.25**
Learning coefficient of output	**0.25**
Epoch	**1052**
Learning algorithm	**Back propagation**
Learning method	**Supervised**
Activation function	**Sigmoidal**

Table 4.2 ANN parameters for the ACS model

The graphical representation of the different trainings and the error in the training is shown in figure 4.7 and 4.8 respectively

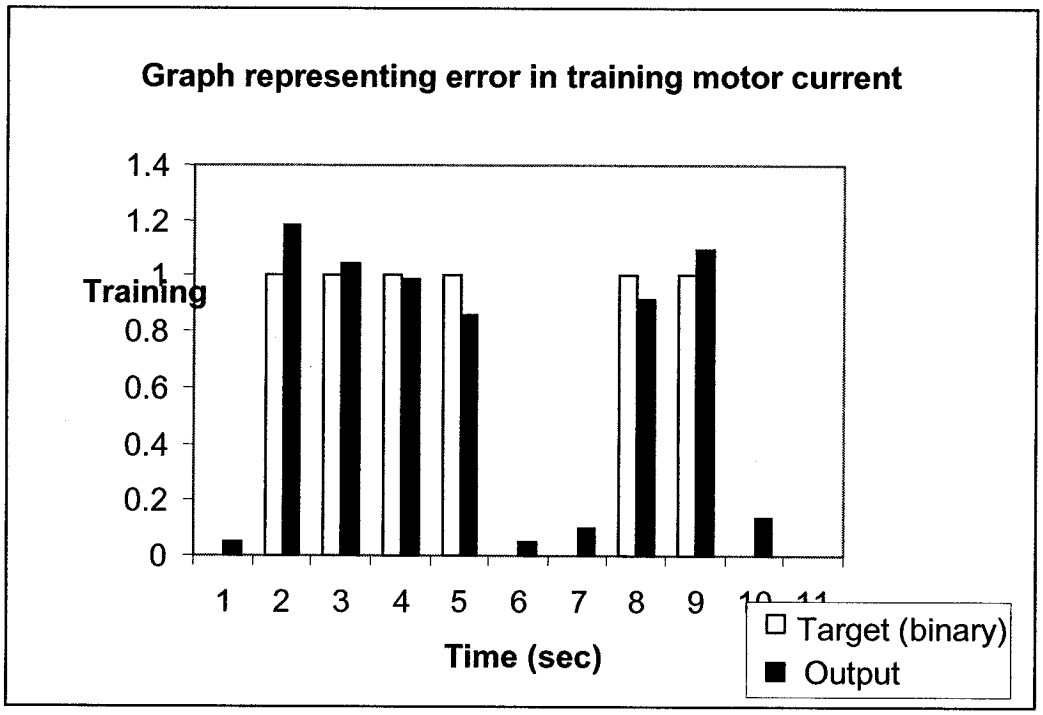


Figure 4.7 Graphical representation of error in training (motor current)

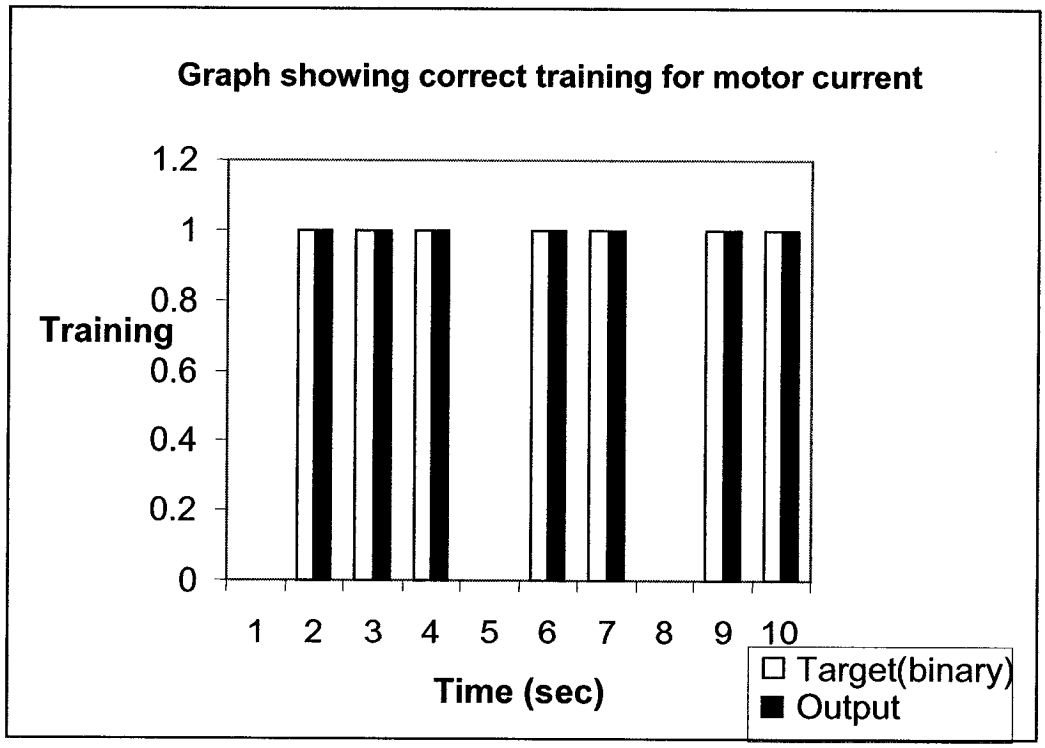


Figure 4.8 Graphical representation of correct training (motor current)

MOTOR CURRENT	ANN MOTOR CURRENT
0.7	0.7
0.528	0.5
0.910	0.9
0.204	0.2

Table 4.3 Testing the ANN with 10 sets of training

The data provided as input was considered as a combination of faulty and non-faulty data and the target was chosen such that it will approximate the input to the nearest non-faulty data. The data after training is compared with the target function to check if the error is minimal. The optimal values of the input variables are also approximated from the output. A sample of the training and optimal value approximation is shown for each of the failure scenarios in Table 4.3 and also graphically in figure 4.9.

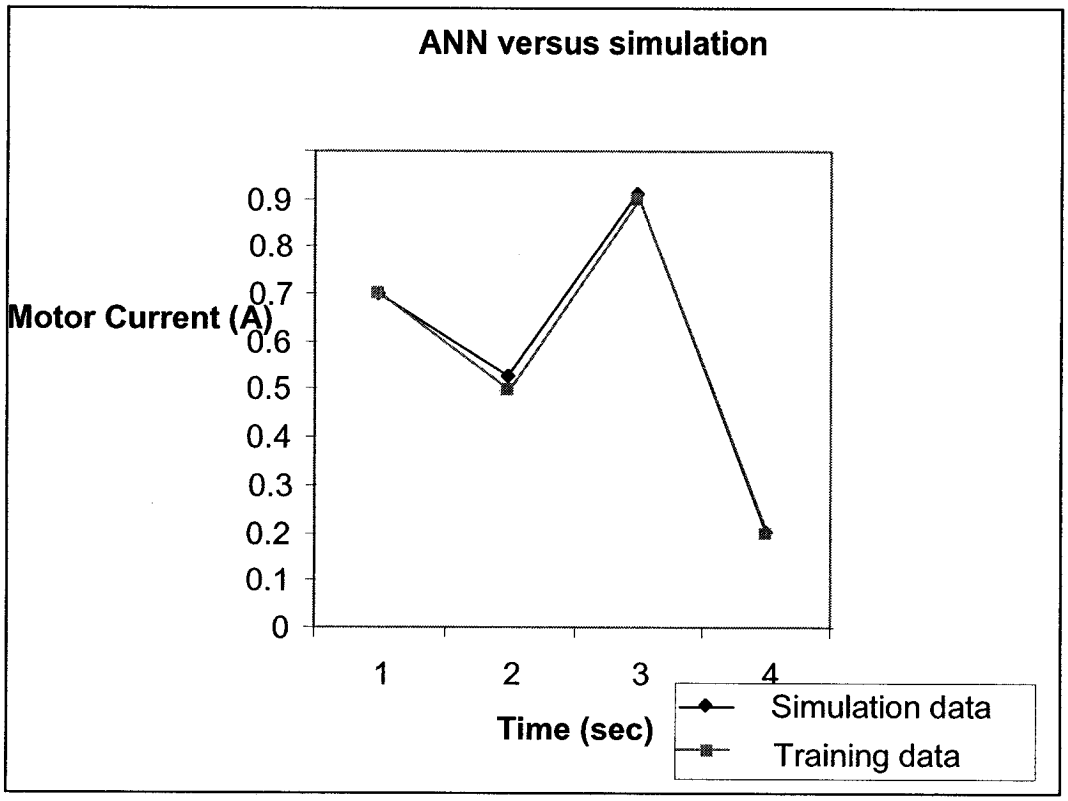


Figure 4.9: ANN versus Simulated Data

This procedure is repeated for the remaining two fault scenarios and the data values of both pitch and bus voltages are also sufficiently trained until the optimal training point is reached. Figures 4.10 and 4.11 show the error in the training and the optimal training respectively. Table 4.4 contains a sample of data which shows the approximation done using the testing data and figure 4.12 shows the graphical form.

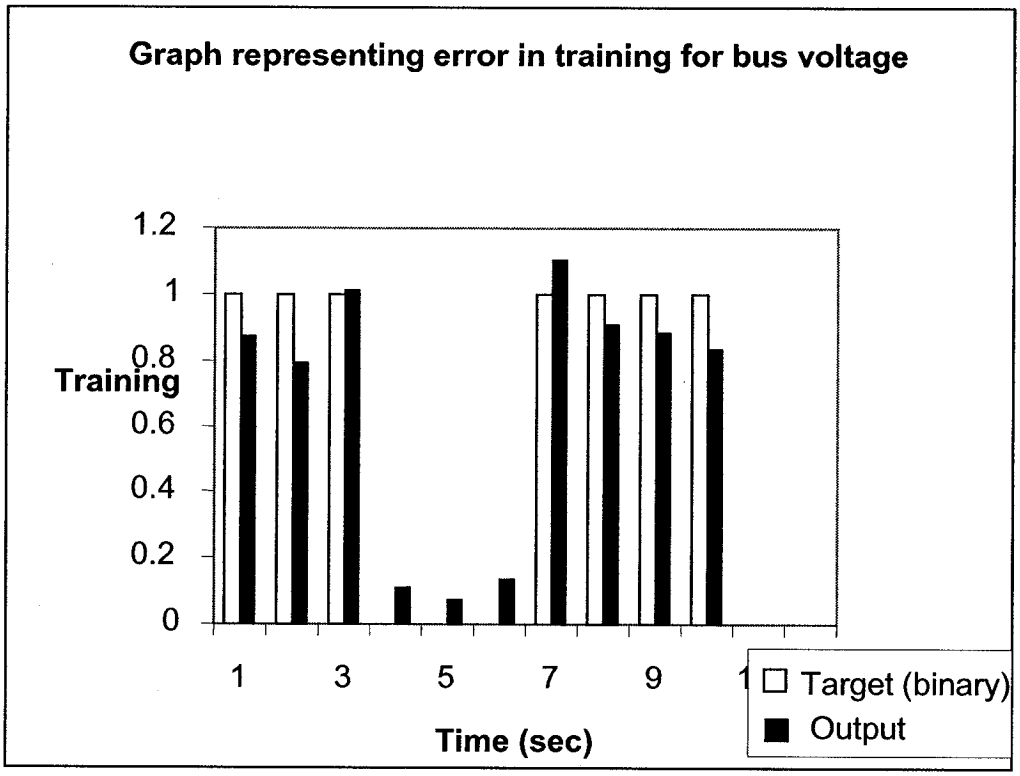


Figure 4.10 Graphical representation of error in training (bus voltage)

BUS VOLTAGE	NEURAL NETWORK BUS VOLTAGE
15.89	15
14.98	15
28.98	28
26.12	26

Table 4.4 Testing the ANN after 10 sets of training

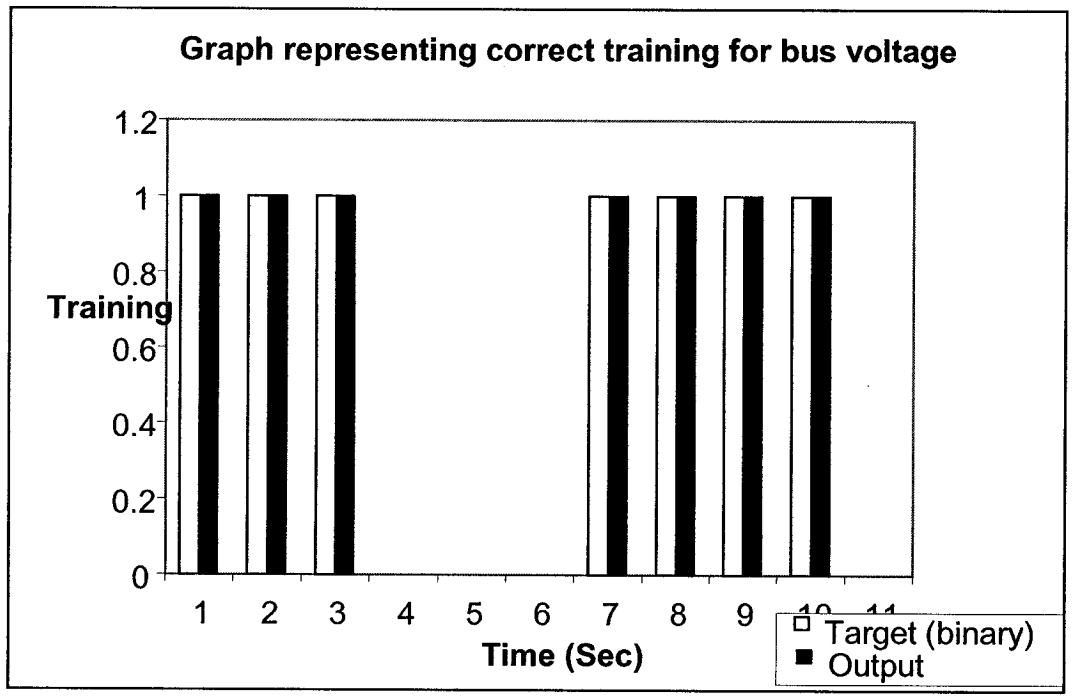


Figure 4.11 Graphical representation of correct Training (Bus Voltage)

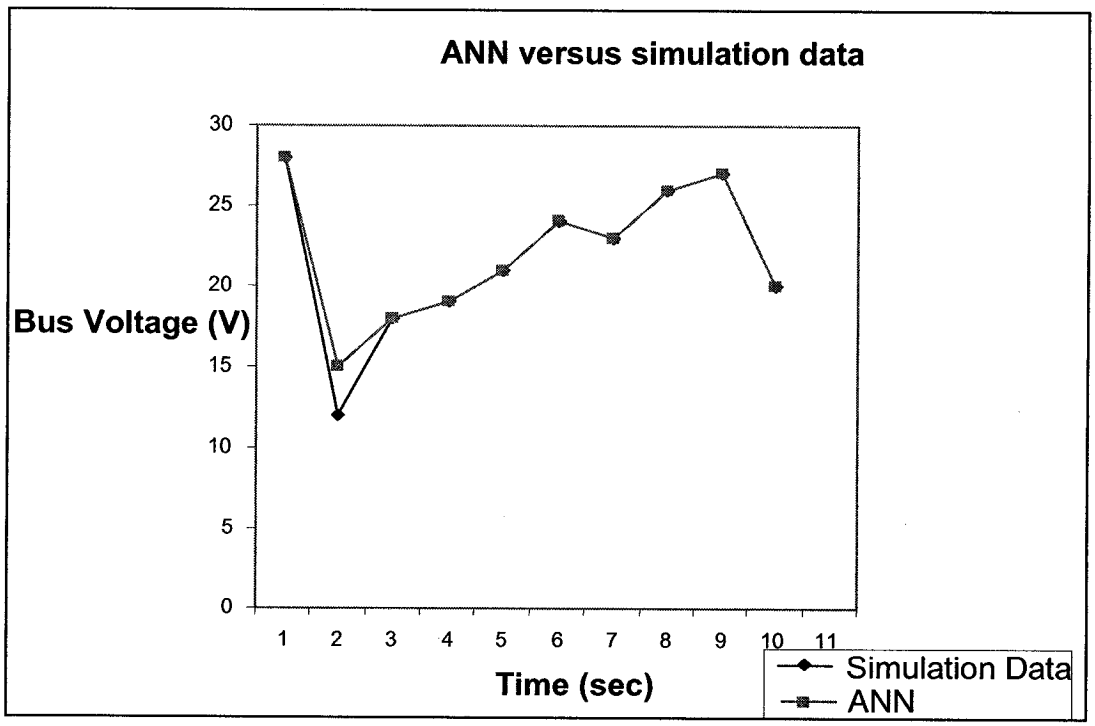


Figure 4.12 ANN versus simulation data

The values that had been approximated by the neural network model for each of

the pitch variables are shown in Table 4.5 below and graphically in figure 4.15. The error in training is shown graphically in figure 4.13 and figure 4.14 as mentioned in the previous cases for other failure scenarios.

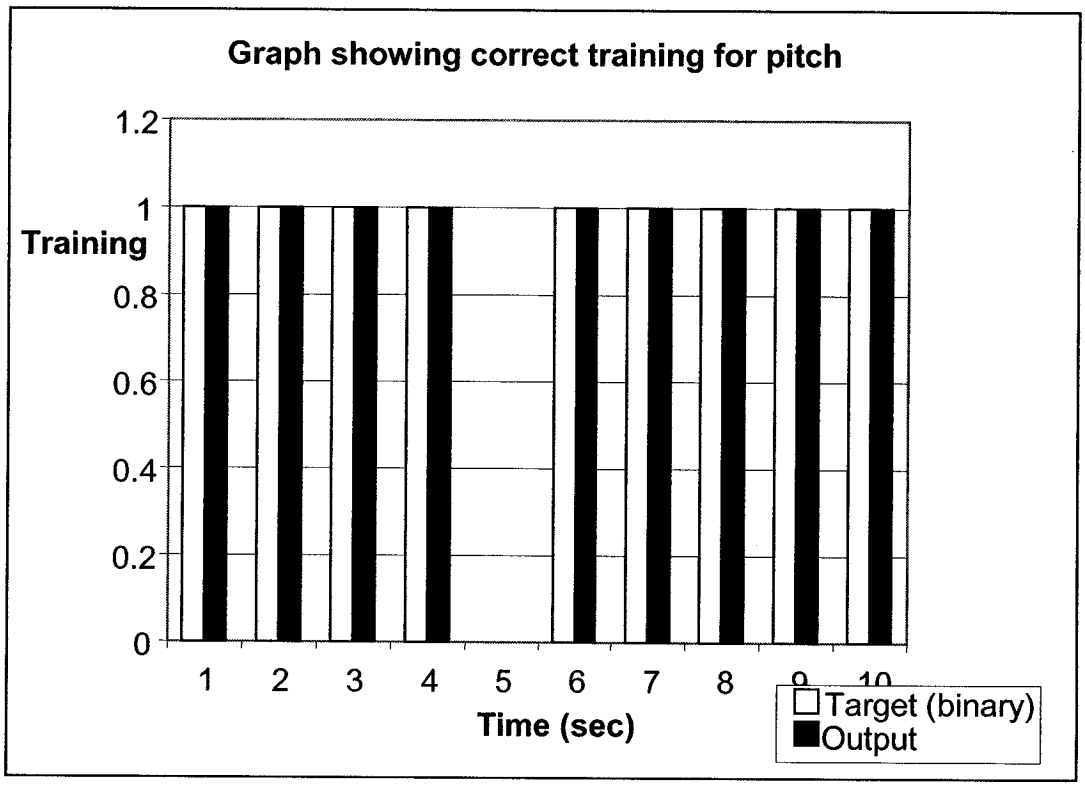


Figure 4.13 Graph showing the correct training (Pitch)

PITCH	ANN PITCH
1.0837	1
2.217	2
0.622	0
1.718	1

Table 4.5 Testing the ANN with 10 sets of training

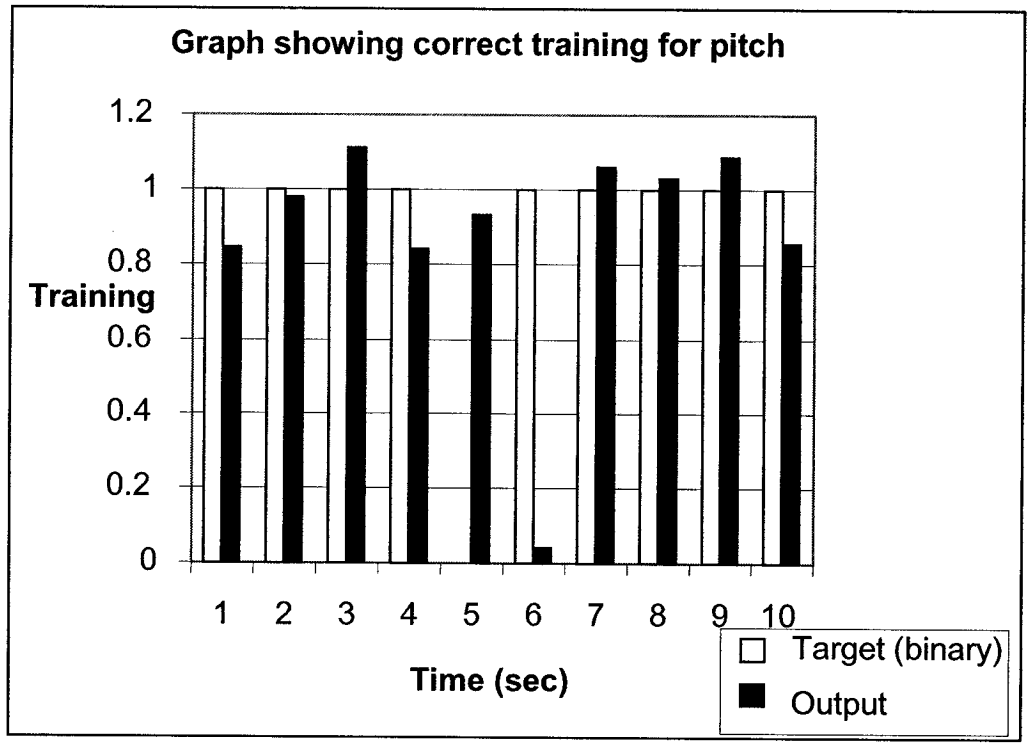


Figure 4.14 Graph showing the error in training (Pitch)

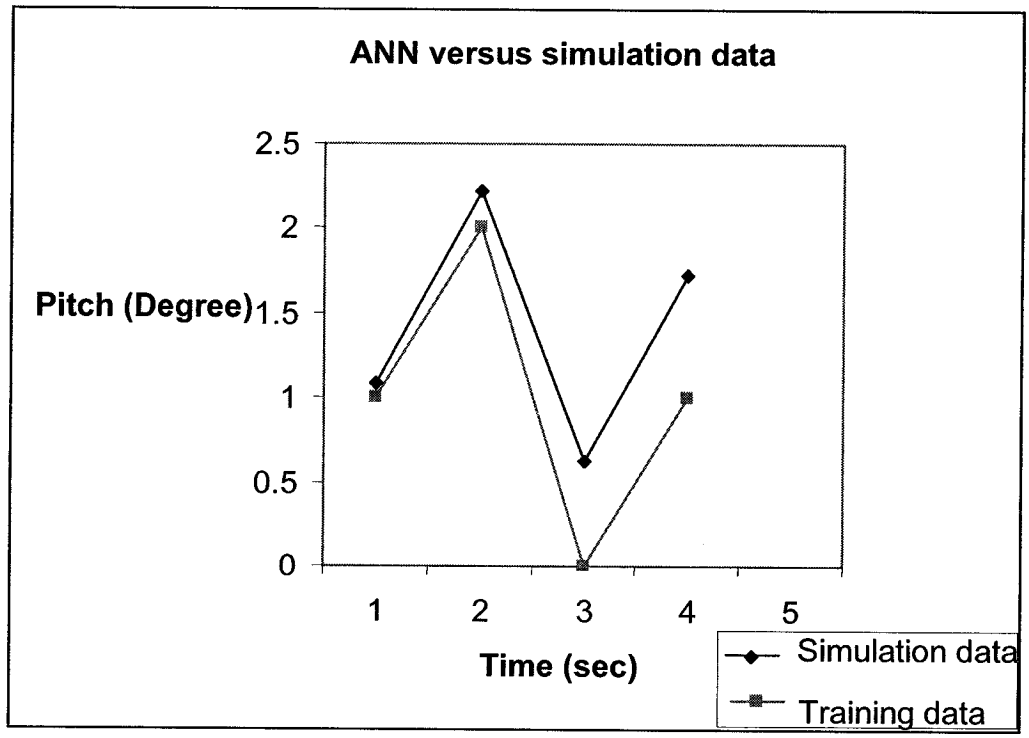


Figure 4.15 NN versus training data

4.3 Rule Extraction Algorithm Applied to Trained Neural Network

The verification of neural network based software systems has been discussed in Section 3.4. It has been established in the literature that although neural networks are highly researched, they cannot be verified using the conventional techniques. The main problems that have been encountered in verifying the neural networks is acquiring an accurate ANN model and then testing it against a real system.

For conventional systems, formal testing can be devised to validate a real system against some model. However, ANNs do not have an adequate model to test against which becomes a limitation [16, 17]. The two levels involved in the validation of the neural network are the training phase and the network validation phase. Training of the neural networks plays an essential role in validation. The process of training can be divided into two main functions [17, 18].

1. Ensuring that the data adequately describes the requirements.
2. Ensuring that the network training process adequately translates the requirements into the expected output.

Validation for neural networks becomes an important and significant issue because of the shortage of high integrity data for most applications [18]. However, network solutions are usually selected for problems that have this data shortage. The types of problems related to data can be a result of noise, imprecision, missing or irrelevant data samples including erroneous measurements. Furthermore, assurance that the data adequately describes the requirements is also needed. Techniques that attempt to extract rules and facts (symbolic reasoning) from the network may be used to fulfill these requirements [18].

Therefore, in order to handle the problems with the data, we had ensured that the Simulink model met with the requirements when the data was gathered from the model. Usually statistical models provide only quantitative data and they do not provide a threshold for safe and unsafe values. Since ACS was a simulated model, the threshold values for each of the data variables under safe and unsafe conditions were first established. Table 4.1 shows the optimum values for the model working without any faults injected.

Realizing verification and validation of neural networks would only be complete if we could extract rules from a trained network and if the extracted rules replicate the workings of the neural network. First, sufficient data was collected successfully from the ACS simulink model under different failure scenarios. It was then divided into three sets which are the training set, testing set and validation set. Once the network had been sufficiently trained the rule extraction algorithms were applied to them. For experimental purposes we had binarised the input and trained the network against the target function. After having chosen the network architecture and the learning algorithm, we had trained the neural network model with all three failure scenarios, after which we had to choose the rule extraction algorithm. At first, we choose the SUBSET algorithm, however there were too many drawbacks. This led to the usage of the VIA rule extraction algorithm as it is classified as a pedagogical technique where there is no changes are required in both the training and learning mechanisms.

4.3.1 Rule Extraction Using SUBSET Algorithm

Having trained the neural network under three different scenarios, now we have to extract rules from them. The SUBSET algorithm is based on a realistic assumption that the levels in the input and output are stored as logical functions corresponding to TRUE or FALSE. Therefore, the activation depends upon the weight values on each of the links connecting the input, hidden and output layers. Rules are generated by finding the subsets of incoming weights where the sum of these weights and bias values makes the activation high enough to generate an output that is equal to TRUE. The rules are of the form

If (Condition) Then (Proposition)

The steps that we followed in order to generate these rules were:

1) Firstly, we identified the positive and negative weights as shown in Table 4.6 which was extracted after training from the neural network model trained for one of the three faulty scenarios. We then identified the greatest positive weight which has the most probable chances of generating an activation which would exceed the threshold such that the output becomes TRUE.

Positive Weight	Negative weight
3.9867	-2.8094
3.7296	-1.6685
2.8842	-0.8924
0.5581	-0.6752
0.5578	-0.5042
0.5475	-0.4590
0.4165	-0.2850
0.3542	-0.1838
0	-0.1787
0	-0.0624

Table 4.6 Weights classified in descending order for positive and negative values

For each set of positive weights P, subsets of these positive weights incoming to the neuron are identified as shown in equation 4.3

$$\sum_{j \in P} (W_{ij} x_i + \theta_i) > 0 \qquad (4.3)$$

where i, j represent the neurons on the input and hidden layers respectively, N is the set of negative weights, x_i is the input that is entering the neuron i, W_{ij} is the weight that is assigned to the link connecting the neurons i and j and θ_i is the bias for the neuron i.

2) The same process was repeated for the negative weights and therefore for each P, a subset N of negative weight links incoming for the neuron i are chosen such that they satisfy the following equation shown in 4.4.

$$(\sum_{j \in P} (W_{ij} x_i) - \sum_{j \in P} (W_{ji} x_j + \theta_i)) > 0 \qquad (4.4)$$

3) For each N state the rule:

"If P and N then the corresponding neuron are selected"

4) Finally, the extensively obvious and subsumed rules are removed. In general, it is advisable to restrict the number of subsets.

The set of rules that were extracted using this algorithm was large therefore we show a single example as a means of explaining the rule extraction method. The resulting network is shown in figure 4.16.

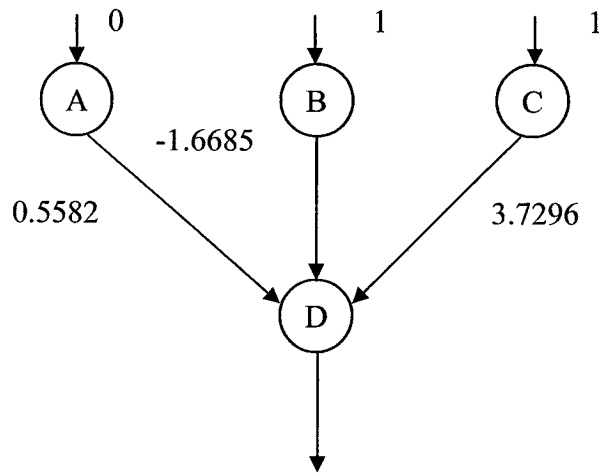


Figure 4.16 Diagram showing initial conditions from which rules are extracted

1. For each output and hidden neuron i, $\sum_{j \in P} (W_{ij}x_i + \theta > 0)$ is checked.

   P1: ((0) (0.5568)-(3.1747))>0 False

   P2: ((1) (-1.6685) + (0.5040))>0 False

   P3: ((1) (3.7296)-(0.7802))>0 True

2. For each of the hidden and output neuron i,

$$(\sum_{j \in P} (W_{ij}x_i) - \sum_{j \in P} (W_{ji}x_j + \theta)) > 0$$

   N1: ((0) (0.5581)-((0.5581) (0)) - 3.1747) > 0 True

N2 :((1) (-1.6685)) - ((-1.6685) + 0.5040)) > 0 False

N3: ((1) (3.7296)-((1)(3.7296)-0.7802)) > 0 True

3. For each N

   If (C and not E) then True

The same procedure is repeated for each of the neurons and the rule base is formed. A large number of rules resulted after removing the overly obvious and subsumed rules for the three faulty variables. For this reason, the rules that were extracted were verified for consistency in a model checker. The percentage of accuracy was considerably less than that of VIA algorithm. Some of the drawbacks of this SUBSET algorithm include that it can be applied with data which is in binary format only and moreover, the set of rules which are extracted is quite large. These drawbacks forced us to consider an alternative algorithm. This brought about the usage of the VIA algorithm for rule extraction.

4.3.2 Rule Extraction using VIA Algorithm

The VIA algorithm is used to extract *if-then* type of rules from a trained neural network. This algorithm is based on a validity interval analysis (VIA), which checks for consistency within a trained neural network. This consistency is achieved by propagating the validity interval through the network in an iterative manner.

The VIA algorithm was directly applied to our model as no new requirements were imposed on the neural network. Thus the data from the ACS simulink model was not binarised for training the neural network, the data was taken directly from the model and the input data provided was taken under faulty scenario as discussed in section 4.1.

The only assumption taken was that the nonlinear transfer function of the neural network must be monotonic and continuous. This is the case with which we trained our neural network and thus no modifications were required. As previously mentioned, the main design behind this algorithm encompasses the notion of a validity interval. Initially, the intervals are randomly assigned then afterwards; they were iteratively refined by removing those intervals which were inconsistent.

The neural network chosen for rule extraction has an input layer, a hidden layer and an output layer. We have considered nine variables that were taken into consideration during the training phase and hence nine neurons are present in the input layer each of which take one variable as input. In the hidden layer, each neuron takes in nine variables namely bus voltage, external torque, motor current, motor torque, wheel speed, torque command voltage, pitch, pitch error, vehicle angular velocity as their input from the nine input layer neurons X_1, $X_2 \ldots X_9$. The network architecture that was chosen is shown in figure 4.5. For simplicity reasons, only three variables containing faults have been considered for training, which are motor current, bus voltage and pitch. Thus the network under consideration has three neurons in the input namely X_1, X_2 and X_3 and hidden layers and three in the output layer having outputs Y_1, Y_2 and Y_3. The network structure is shown in Figure 4.17 and the weight and bias information that are extracted from the network are shown in the Table 4.7.

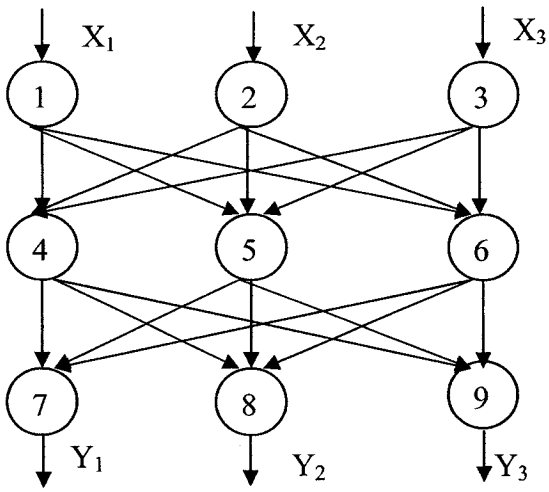


Figure 4.17 Network topology of the neuron model under consideration

From node	To node		
	Hidden4	Hidden5	Hidden6
1(input)	1.4111	-2.6423	2.8074
2(input)	-1.7672	1.8327	3.0807
3(input)	1.0559	-3.1011	2.3757
Bias	-0.1716	0.0913	0.0746
	Output1	Output2	Output3
4(Hidden)	0.2154	0.0268	0.3705
5(Hidden)	-0.3706	0.4040	-0.1404
6(Hidden)	-0.0340	-0.6008	0.1595
Bias	-0.3726	0.1374	-0.1504

Table 4.7 Weights and bias neuron model shown in figure 4.17

Once the network is well-trained, the weight and bias information are extracted from the network and the activation function is calculated. Since we assume the network is sufficiently trained, we consider static values of both the weights and biases.

The VIA algorithm has two phases, namely the forward and backward phases. The forward phase consists of when the intervals of activations are propagated in the forward direction. We randomly set a maximum range on the initial activation values which are the validity intervals of the neuron for the forward phase. We used linear programming and simplex algorithm to optimize these intervals in the backward phase. Inconsistent activation values are removed during the refinement using linear programming. All the activation values consistent with the initial interval are also consistent with the refined interval. Thus the rule extraction has two main steps.

1. Finding rule hypotheses in the form of validity intervals for neurons of the neural network
2. Using VIA to verify these initial intervals and refine them by detecting and removing all of the inconsistencies.

The first part involves finding the rule hypotheses in the form of validity intervals for neurons. The algorithm proposes both search based and learning based mechanisms. We have a discrete data set and therefore, the search based technique is the better option. A search tree was constructed with the most general rule at the root of the tree. Each node in the tree spans a subtree of a more specific rule. The VIA algorithm is applied to the node. If the rule is verified the subgraph is removed.

In order to derive the general rule we first calculate the initial activation values. The activation value is constrained such that the successor receives the value of net_i which is within the validity interval. The activation x_i is computed in two steps in equation 4.5 and 4.6:

$$net_i = \sum_{j \in P} (W_{ij} x_i + \theta_i) \qquad (4.5)$$

$$x_i = \sigma_i(net_i) \qquad (4.6)$$

Here the net_i is called the net-input and W_{ij} and θ_i are weight and bias respectively. σ_i denotes the transfer function which it infers from the training of the back-propagation algorithm as shown in equation 4.7.

$$\sigma_i(net_i) = \frac{1}{1+e^{-neti}} \quad \text{with } \sigma_i^{-1}(x_i) = -\ln(\frac{1}{x_i} - 1) \qquad (4.7)$$

Once the validity interval is extracted and propagated through the network the activation function of the successor units are smaller than the previous values for all consistent links.

Once the input and target are given and the neural network is trained sufficiently, the weights and bias information are extracted. For the purpose of analysis we have considered three parameters in which faults were injected as discussed in section 4.1. The positive and negative values of the weights of the output and hidden units are first classified in descending order in two sets as previously shown in Table 4.7.

Within the algorithm, for each hidden or output unit j the algorithm starts from the most positive weight value(say i) and then it searches for the incoming link that causes the node in the next layer to be active regardless of other input links coming to that node

(from the other neurons of the input layer). If such nodes are identified then they are written as a rule. The intermediate rules extracted are of the form

$$If\,[(w_1X_1+w_2X_2+w_3X_3+\ldots\ldots+w_nX_n) >= \theta+\Delta] \xrightarrow{C_f} consequent$$

where w refers to the weights, x refers to the input, θ refers to the bias value and C_f is the certainty factor which represents the measure of the belief in the extracted rules. The Δ refers to the certainty factor. The certainty factor for the sigmoidal function is calculated using the equation 4.8.

$$C_f = \begin{cases} \dfrac{1}{1+\exp^{\left(\sum_{i=1}^{n} W_{ij}x_i - \theta_j - \Delta\right)}} & \text{If the activation is sigmoidal} \end{cases} \qquad (4.7)$$

Note that a range of x_i values may satisfy an intermediate rule and thus it is desirable to determine a suitable extreme value within the range. To make this traceable, each input range is discretized into a small number of values that can subsequently be examined. Thus, each input feature $x_i \in (a_i, b_i)$ is discretized using k intervals as shown in equation 4.8.

$$(Di \in \{di_{10} = \text{ai}, \text{d}_{\text{i},1}, \ldots\ldots\ldots\ d_{i,k-1}, d_{i,k} = b_i\}) \qquad (4.8)$$

where $d_{i,l}$ and $d_{i,l-1}$ are the upper and lower boundary values of interval l of input x_i.

Through this method, the intermediate rules are first determined for our network. Then, the positive links for each of the three variables were considered in order to determine the general rules. From these rules, a hypothesis about the problem is considered. This can be proved or disproved using prior knowledge of the model and the rules previously generated. The hypotheses of the problem involved in our model are as follows:

Motor_Current ∈ {True, False}

Pitch $\in$ {True, False}

Bus_Voltage $\in$ {True, False}

Within these hypotheses, all of these variables can be either faulty or non-faulty which is denoted as True or False respectively. Essentially we are attempting to realize a target concept or a particular goal within our network. The extracted rules would be deemed correct if they prove this target.

The target within our system is as follows:

- Problem M1 : If pitch=True, Motor_current=True and Bus_voltage=True Then M1(Status=SAFE)

As mentioned earlier, we have considered faults within three variables and thus the other variables are set to TRUE by default. There is no checking necessary for these statements.

- Problem M2: If any of the three variables has a value other than TRUE Then M2(Status=UNSAFE)

After having stated our hypothesis and the target function which we are trying to realize, we then extract the general rule by finding out the activation function of the most positive weight which exceeds the threshold imposed by that neuron. This is done by using a Directed Acrylic Graph as shown in figure 4.18. The list of consistent rules that resulted was large and therefore we had to impose linear constraints on the network in order to get a finite set of consistent rules. These constraints were only imposed in the case where we had all nine variables present. Within our concise example of three variables, these constraints were not required. The constraints imposed were the following:

-If sum of all feature values >1 then not M1

-If sum of all feature values were <=1 then M2

As a result, we iteratively removed all the inconsistent links within our network. As stated earlier, the target function is realizing whether the system is safe or unsafe. Some of the rules that were extracted in the process of realizing the target function are shown in Table 4.8.

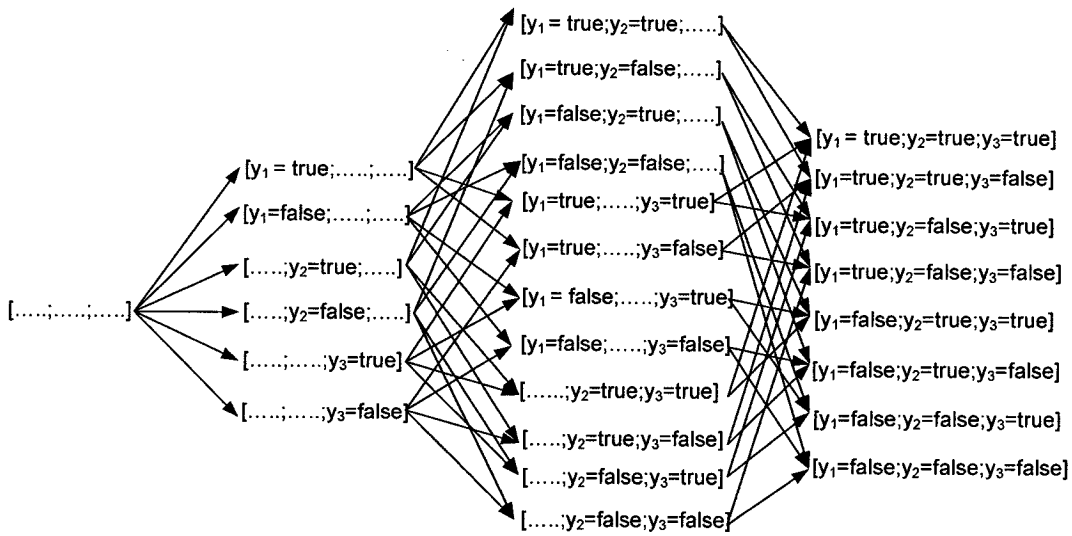
Pitch	Motor Current	Bus Voltage	Class
True	False	False	Not M1
False	True	True	Not M1
False	False	True	Not M1
False	False	False	Not M1
True	True	False	Not M1
False	True	True	Not M1
True	False	True	Not M1
True	True	True	M1

Table 4.8 The extracted rules from the network learned

After the general rule was established the Directed Acrylic rule graph (DAG) was constructed for the three fault variables which we have taken under consideration. This enabled us to prove or disprove the hypotheses and remove the inconsistent rules from the finite search space.

Within this DAG, a breadth first search is done to extract the rules by knowing the

target concept. The entire search space consists of all the possible rules. The root forms the most general rule where the status of our three variables (true or false) is unknown. The search then progresses and each of the rules are either proved or disproved. It can be seen that if all three variables are true then the target concept is proved and the system is deemed safe. Thus once the general rule is proved correct the entire subgraph belongs to the same class. Thus the DAG will generate the final rules which are shown in Table 4.8 Therefore, the VIA analysis has been successful in generating the proper rules thereby proving our target concept.



4.18 Directed Acrylic Graph for rules with three Boolean variables[18]

The main advantages of using the rule extraction algorithms is better understanding of the output of the neural network and a higher speed of learning. A pedagogical approach to the problem was used and the symbolic information that has been extracted from the rules is fed back into the network. These extracted rules are of

the Boolean propositional logic (If-then form) form which gives the relationship between the inputs, weights and the output activation. By looking at the output activation functions a distinct pattern can be realized which is unique for the faulty data and non faulty inputs. The extracted rules for each individual neuron are aggregated over the entire network to determine a global rule set. These extracted rules can then be used to generate a specification at training *a posteriori.*

As a result, the rule extraction was a process of translating the neural network into a straightforward comprehensible format which allowed us to understand the inner functioning of the neural network.

There are a number of factors that determine the accuracy of rule extraction algorithms. These include fidelity, quality and comprehensibility of the extracted rules. Fidelity is the ability of the extracted rules to mimic the working of the neural networks. Comprehensibility is the ability of general users to understand the rules with ease. The rule extraction algorithms used, their classification and their applicability is shown in Table 4.9. The quality of the rules and the complexity of the two algorithms that we have utilized are shown in Table 4.10

Rule extraction algorithm	Classification	Applicability	
SUBSET	Decompositional	3-Layer feed forward, backpropagation with weights	Binary
VIA	Pedagogical	3-layer feed forward, backpropagation with weights	Discrete

Table 4.9 Classification and applicability

Complexity			Quality of Rules	
Dependence	Evaluation	Accuracy	Fidelity	Comprehensibility
Number of rules are more	High	Low	High	High
Number of neurons and layers	High	High	Low	low

Table 4.10 Quality of rules and complexity

4.4 Formal Specification of the Neural Network in UPPAAL

Formally specifying the neural network that was designed based on the extracted rules is the first step taken towards formal verification of the neural network. The SUBSET algorithm was applied to the trained neural network and symbolic rules were extracted. The rules were checked for correctness by representing them in a model checker such as UPPAAL and then checked against the target. The entire rule extraction phase of our proposed model is modeled in UPPAAL and the verification is done using the model checker for various safety and liveness properties. Each layer of the neural network is modeled as a finite state machine synchronized with each other through message exchanges over communicating channels.

Our design contains a main machine as shown in figure 4.19 which is synchronously running with the finite state machine of each individual node in the process. This state machine moves from the SAFE state to the UNSAFE state depending on the input and target values. In addition, we have three different processes representing our Input layer, Hidden layer and Output layer. The Neural network training and rule extraction from trained neural network is shown in detail in section 4.2 and section 4.3.

Section 4.3.1 in specific focuses on rule extraction using SUBSET algorithm from trained neural network. The extracted rules are then model checked using the model checking capabilities of the UPPAAL tool. Having trained the neural network model in supervised training algorithms, we already know the expected target and the output that we derived using the neural network model designed for training. The rule extraction algorithm further provided us a comparison of the target output and the output that was obtained. The target and the output obtained are further checked for correctness using the model checking capabilities of UPPAAL.

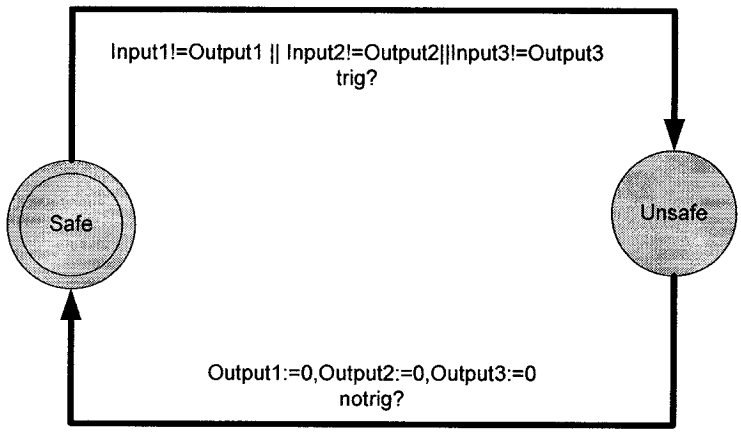


Figure 4.19 Main finite State machine in UPPAAL

Each layer within the neural network structure is represented by a separate finite state machine which is then synchronized with the main machine. Each state machine is designed such that based on the given values for *Activation* and *Threshold* a particular node is reached. After comparing the *Activation* and *Threshold* values, a decision is made and one of two states is attained namely triggered or not triggered. In figure 4.20, the first node represents the input layer. If the Activation is greater than the threshold

then the neuron is triggered and goes to the Trigger_A, else the neuron A is not triggered and it goes to the Do_Not_Trigger_A state. The channel *trig and notrig* synchronizes this state machine with the main machine. Thus similar to the working of the rule extraction algorithm, our model is designed to make decisions on if the neuron A is triggered or not.

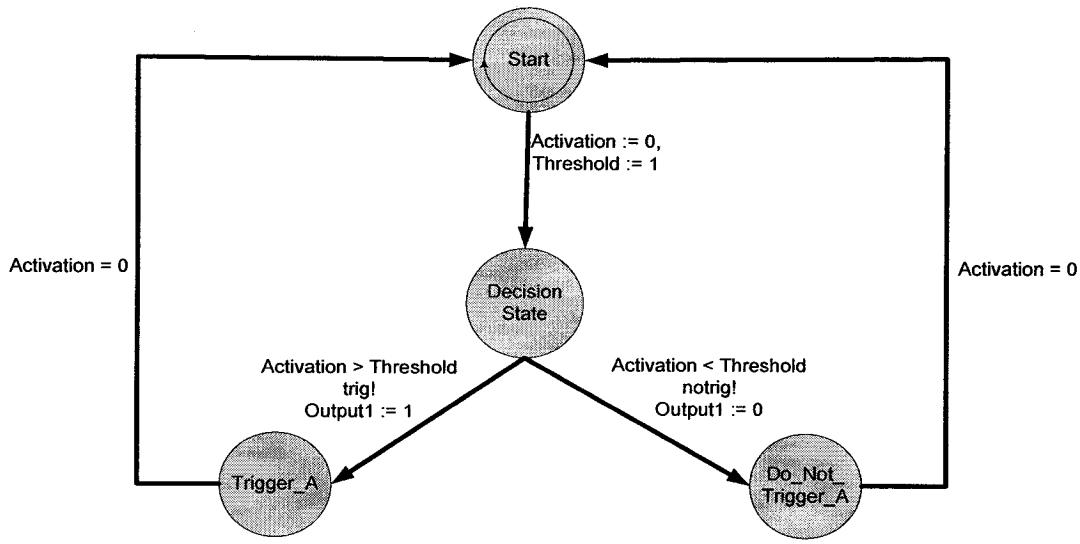


Figure 4.20 Input layer state machines in UPPAAL

In figure 4.21, the first node represents the hidden layer and has a similar structure. It is triggered on the basis of the second activation function and the second threshold function. This is also synchronized with the main machine using the *trig* and *notrig* channels.

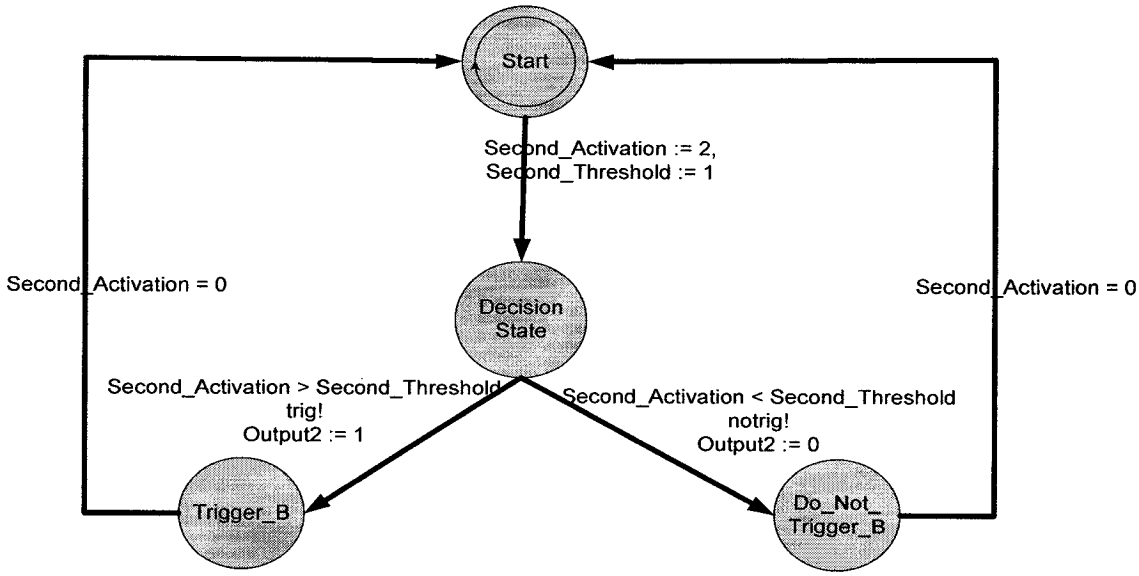


Figure 4.21 Hidden layer finite state machines in UPPAAL

In figure 4.22, the first node represents the output layer. This machine is similar to the input and hidden layers and is also synchronized with the main machine using the *trig* and *notrig* channels.

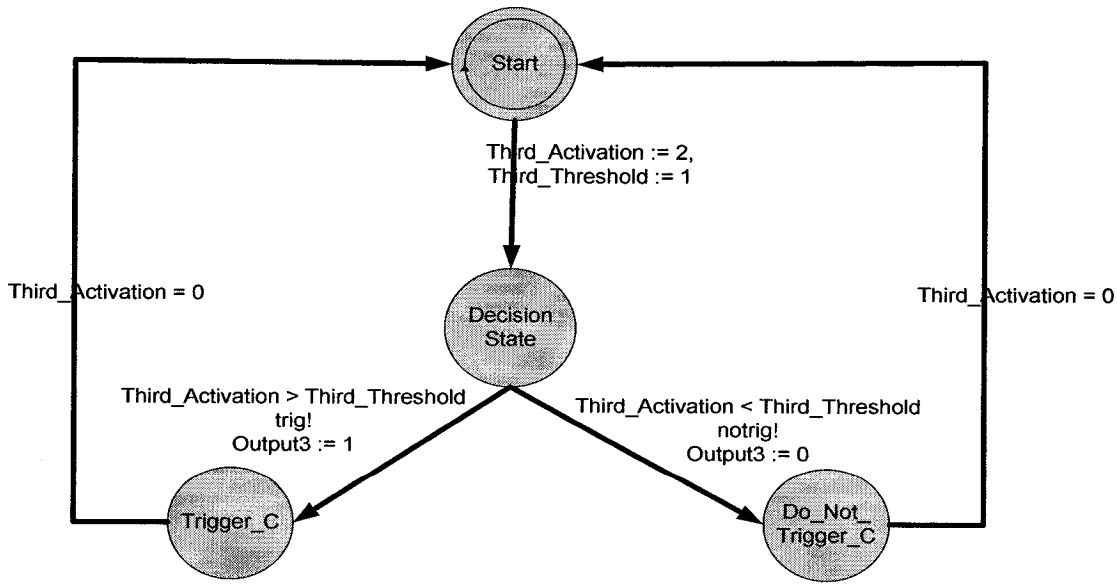


Figure 4.22: Output layer finite State machine in UPPAAL

4.4.1 Simulation and Verification

The model has been simulated using the UPPAAL simulator. Our Target value is set to (0,1,1) which states that neuron A was not triggered and neuron B and neuron C were triggered. The output resulting from our model check, should be the same. If the expected target and the output that we had derived are the same then the system goes to the *Safe* state else the system goes to the *Unsafe* state which is shown in the simulation results using the message sequence diagram. It can be seen from the message sequence diagram in figure 4.23 that when the expected output and target are different the system goes to the *Unsafe* state else it remains in the *Safe* state. The results for a single simulation are shown however the simulation is repeated for all three fault scenarios. The same process is repeated for each output and the neural network model is checked for correctness.

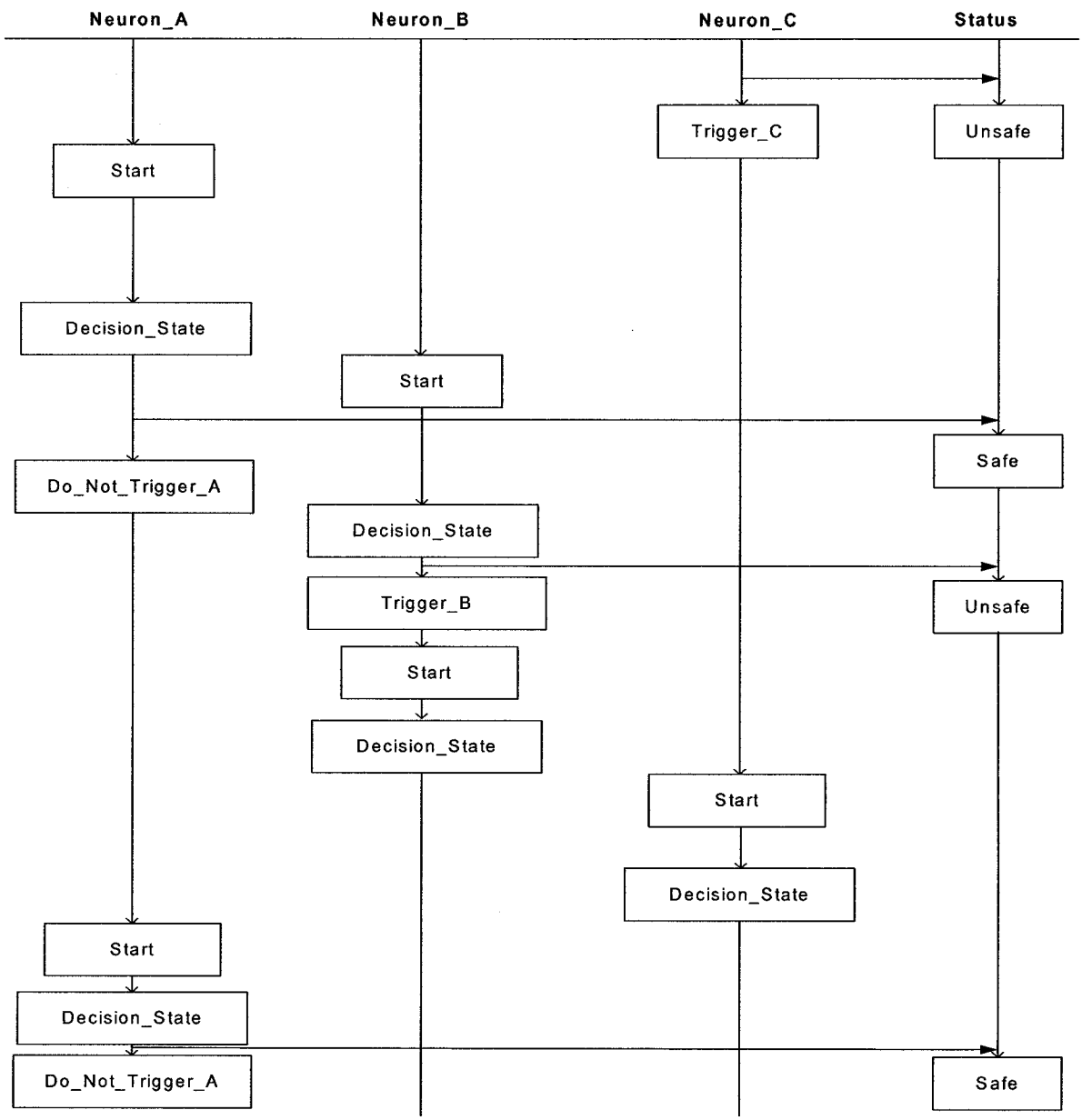


Figure 4.23: Simulation results.

In figure 4.24 we show the model checked results for certain liveness and safety properties. While conducting this simulation, the input value was taken as (0, 1, 1). This means that the neuron A is not triggered. With respect to the first state machine, the state

Trigger_A is reached and thus the model checker shows property not satisfied. The system is also checked for deadlock which does not exist within the system therefore the property is satisfied. The results of the reachability analysis performed are shown in figure 4.25. In figure 4.25 we have shown that the finite state machine that was created and synchronized in UPPAAL does not have any deadlock. Various other properties such as the reachability analysis has also been shown, except the first property which is not be satisfied because the output expected is (0, 1, 1) and thus the Trigger_A state is not activated and hence when we check to see if that state is reached that property is not satisfied. All other properties such as Trigger_B and Trigger_C are satisfied because those states are activated. Hence we have shown that the model checking capabilities of UPPAAL can be used for the verification of neural network based systems to be used in safety critical systems using formal methods.

States	States Reachable	Status
P0	**E<>A.Trigger_A**	**Not Satisfied**
P1	**E<>C.Trigger_C**	**Satisfied**
P2	**E<>B.Trigger_B**	**Satisfied**
P3	**E<>A.Do_Not_Trigger_A**	**Satisfied**
P4	**E<>Status.Unsafe**	**Satisfied**
P5	**E<> Status.Safe**	**Satisfied**
P6	**A[] not Deadlock**	**Satisfied**

Figure 4.11 Model checking and reachability analysis of the system in UPAAL

4.5 Formal Specification in PVS

The rules extracted from the VIA algorithm are also checked for correctness against the ACS model specification. The extracted rules are formally specified using a theorem prover called PVS. Each of the nine variables given as an input to the neural network model is checked for correctness. The variables are set to either TRUE or FALSE indicated by 0 or 1 in the specification. The theorems are then checked and the system is deemed safe when all of the variables result in a TRUE value. Otherwise, the system is considered unsafe. The variables are set to TRUE only when the faults are tolerated in the initial training phase of the neural network and the rules extracted are replicating the ACS model completely as is the case within the actual neural network structure. It is indicated in our PVS code by the first eight AXIOMS that even if one of the nine variables are FALSE then the system's status is unsafe. The theorem proving capabilities of PVS proves that the system is deemed safe only when all of the variables are true and this would happen only when the initial specification and the extracted rules are similar which in other words means that the neural network behavior is completely replicated. The rules extracted using VIA algorithm was similar to the initial specification so we had proved using theorem prover that the system was proven to be safe.

Hence formal verification has proved that V&V of neural networks is similar to the conventional verification if a well specified model and adequate data is available for training. We have shown that neural networks can be verified through a proposed framework for use in safety critical systems and an understanding of the working of the neural networks can be achieved. The same formal verification could have been done in

a model checker such as UPPAAL also, but we wanted to show that different formal techniques can be used for V&V of neural networks. V&V of ANN was actually to prove that the neural networks were indeed behaving in the way it was intended to behave and there is no incomprehensibility associated with the neural network training. Neural networks can be used in safety critical systems only when they clear a certification procedure and for them to be able to clear this process they must be able to prove that though neural networks are adaptive, they are indeed adapting to the correct data and the neural network training is indeed correct. This is done by extracting the rules from the trained neural networks and formally verifying those using formal methods to see if the extracted rules replicate the specified model. Thus the proposed framework was successful in showing this for the three failure scenarios for which the neural networks was trained and the rules were extracted from them and tested formally.

```
simplecheck  % [ parameters ]
               : THEORY

  BEGIN
mybool: TYPE+=int
Pitch: int=0
Motor_Current: int=0
Motor_Torque: int=0
Pitch_Error: int=0
Vehicle_Velocity:int=0
Tcommand_Voltage:int=0
Wheel_Speed:int=0
B_Voltage:int=0
External_Torque:int=0
Status: int


Rule: AXIOM ((Pitch=1)AND
(Motor_Current=0)AND(Motor_Torque=0)AND(Pitch_Error=0) AND
(Vehicle_Velocity=0) AND (Tcommand_Voltage=0) AND (Wheel_Speed=0) AND
(Bus_Voltage=0) AND (External_Torque=0)) IMPLIES (Status=1)

Rule1: AXIOM ((Pitch_Error =1) AND (Motor_Current =0) AND (Motor_Torque
=0) AND (Pitch=0) AND (Vehicle_Velocity=0) AND (Tcommand_Voltage=0) AND
```

```
(Wheel_Speed=0) AND (Bus_Voltage=0) AND (External_Torque=0)) IMPLIES
(Status=1)

Rule2: AXIOM ((Motor_Current=1) AND (Pitch_Error=0) AND
(Motor_Torque=0) AND (Pitch=0) AND (Vehicle_Velocity=0) AND
(Tcommand_Voltage=0) AND (Wheel_Speed=0) AND (Bus_Voltage=0) AND
(External_Torque=0)) IMPLIES (Status=1)

Rule3: AXIOM ((Motor_Torque=1) AND (Pitch_Error=0) AND
(Motor_Current=0) AND (Pitch=0) AND (Vehicle_Velocity=0) AND
(Tcommand_Voltage=0) AND (Wheel_Speed=0) AND (Bus_Voltage=0) AND
(External_Torque=0)) IMPLIES (Status=1)

Rule4: AXIOM ((Vehicle_Vehicle=1) AND (Pitch_Error=0) AND
(Motor_Current=0) AND (Pitch=0) AND (Motor_Torque=0) AND
(Tcommand_Voltage=0) AND (Wheel_Speed=0) AND (Bus_Voltage=0) AND
(External_Torque=0)) IMPLIES (Status=1)

Rule5: AXIOM ((Tcommand_Voltage=1) AND (Pitch_Error=0) AND
(Motor_Current=0) AND (Pitch=0) AND (Motor_Torque=0) AND
(Vehicle_Velocity=0) AND (Wheel_Speed=0) AND (Bus_Voltage=0) AND
(External_Torque=0)) IMPLIES (Status=1)

Rule6: AXIOM ((Wheel_Speed=1) AND (Pitch_Error=0) AND (Motor_Current=0)
AND (Pitch=0) AND (Motor_Torque=0) AND (Tcommand_Voltage=0) AND
(Vehicle_Velocity=0) AND (Bus_Voltage=0) AND (External_Torque=0))
IMPLIES (Status=1)

Rule7: AXIOM ((External_Torque=1) AND(Pitch_Error=0) AND
(Motor_Curent=0) AND (Pitch=0) AND (Motor_Torque=0) AND
(Tcommand_Voltage=0) AND (Wheel_Speed=0) AND (Bus_Voltage=0) AND
(Vehicle_Velocity=0)) IMPLIES (Status=1)

Rule8: AXIOM ((Bus_Voltage=1) AND (Pitch_Error=0) AND (Motor_Current=0)
AND (Pitch=0) AND (Motor_Torque=0) AND (Tcommand_Voltage=0) AND
(Wheel_Speed=0) AND (External_Torque=0) AND (Vehicle_Velocity=0))
IMPLIES (Status=1)



FINAL: THEOREM (Motor_Current=0    AND Motor_Torque=0    AND Pitch=0
AND Pitch_Error=0    AND Vehicle_Velocity=0    AND Tcommand_Voltage=0
AND Wheel_Speed=0    AND Bus_Voltage=0    AND External_Torque=0    )
IMPLIES Status=0
FINAL1: THEOREM (Motor_Current=0    AND Motor_Torque=0    AND Pitch=0
AND Pitch_Error=0    AND Vehicle_Velocity=0    AND Tcommand_Voltage=0
AND Wheel_Speed=0    AND Bus_Voltage=0    AND External_Torque=1    )
IMPLIES Status=1

  END simplecheck
```

4.6 Conclusion

In this chapter we have shown the results of training the data from the ACS simulink model using neural networks. We have shown the developed neural network model which was trained in such way as to not allow overtraining and we used a generate and test procedure during the training of the neural network where the training data was divided into training set, testing set and validation set. The training was successfully completed and rules are extracted from the trained neural network. The extracted rules are then checked for correctness with the initial specification using formal techniques such as a model checker and a theorem prover. We had chosen two different rule extraction algorithms. The rules extracted from the SUBSET algorithm was checked for correctness against the specification of the ACS model using the UPPAAL model checker. We realized that the rules extracted from the SUBSET algorithm did not really replicate the specification of the ACS model. Some of the reasons why SUBSET algorithm did not work too well for our model were that the number of the rules that were extracted for our problem was so many that when overly obvious rules were subsumed rules was removed. The removed rules would have hidden a significant structure of the neural model. The other reasons was that there is always only a certain level to which we can safely restrict the number of antecedent, as there is always a tradeoff between the number of rules that were extracted and the accurate reproduction of the network's behavior. We understood that the rule extraction algorithm and the formal verification tool that is chosen are subjective to the problem in hand. That is the reason that SUBSET algorithms, though being established as an algorithm from which rules have been extracted successfully was not able to completely replicate the network's behavior

successfully in our problem. We have shown rule extraction from SUBSET algorithm only under one faulty scenario as the rules extracted were so many taking into account one scenario itself. SUBSET algorithm only accepts data in the binary format. The algorithm was not successful in replicating the network's behavior under one fault scenario itself. The rules extracted from SUBSET algorithm was successfully verified using a model checker. Though the rules were not consistent with the model, it was shown that a model checker could check for correctness of the specified model. UPPAAL was successful in showing that the system was considered unsafe as the extracted rules were not correct. Due to the fact that the SUBSET algorithm did not provide satisfactory results, we advanced our work to include the VIA algorithm for rule extraction. VIA algorithm can be directly applied to the model as no new requirements were imposed. Rules were extracted from the neural network model trained using a simple back propagation algorithm. The neural model was trained under all three failure scenarios and the extracted rules replicated the network's behavior to a good accuracy level. The results were that the VIA algorithm returned accurate results with respect to our given model. In order to illustrate that formal methods can be used within trained neural networks we chose an alternative correctness checker known as the PVS theorem prover. The theorem prover helped us realize that the rules extracted from the VIA algorithm replicated the initial specification. So we have realized that for the proposed approach to work properly three different criterion should be satisfied (1) There should be adequate data for training the neural network in the initial training phase (2) The Rule extraction algorithm used is specific to the domain that has been chosen (3) Formal verification does play an important role in the verification and validation of the neural

network based software systems used in safety critical systems.

Chapter 5

Conclusions and Future Work

As time progresses, autonomous systems evolve and thus, through discreet learning and intelligent decision capabilities, they are able to respond to situations that have not been a priori identified or analyzed. In particular, neural network-based systems have emerged as a powerful class of autonomous and adaptive systems. Due to their learning abilities, ANN have been increasingly attracting attention in applications where autonomy is critical and where the pre-identification of the possible fault scenarios is not extensive. Though ANN have been accepted and used in fault diagnosis, there are some specific concerns with respect to their reliability and acceptability in safety critical applications, since the occurrence of faults may lead to catastrophic effects in the system. For this reason, ANN must undergo a certification procedure in order to be accepted for use in safety critical applications. Thus, it was established that for ANN to be accepted into the safety critical domain, they need to be verified and validated thoroughly to prove that the model is consistent with the specification, and thus, formal analysis was considered. One formal method used for analyzing neural networks was symbolic rule extraction from trained neural networks.

In this thesis, an approach for verification and validation of neural network-based software systems has been proposed. This approach aims to provide a framework for formal verification of neural networks, which allows them to be successfully used in safety critical autonomous systems. The input for our approach involves data collected from an attitude control subsystem of a satellite model which was modeled in Simulink

[32]. The model was used to generate simulated system data under faulty and non-faulty system conditions, which is an essential part of the fault-diagnosis study performed in [32]. Faults were injected into the model and this resulted in three different fault scenarios used within our framework. In this thesis, neural networks have been used as a tool for diagnosing faults in the Attitude Control Subsystem of a satellite. The framework proposed in this thesis utilizes rule extraction without which the explanation capability of the working of the neural networks could not have been realized.

In order to support the proposed fault-diagnosis approach, a neural network model was developed which was trained with faulty data and tested against non-faulty data to see if the network behaved in a non-faulty manner. The neural networks confine the need for a detailed knowledge of the design and construction of the system under consideration. To ascertain that the neural network is functional, the rules they follow must be extracted. In our approach, an existing rule extraction algorithm was modified and adapted for extracting rules from a trained neural network. The advantage of this rule extraction algorithm is that the algorithm does not require any special type of training topology to be established.

Given the system parameters under faulty and fault-free conditions, the neural model generated was able to diagnose the faulty and non-faulty data. The rule extraction algorithm further extracted the rules and determined if the neural model was functional. At this point, the specified system could be classified as safe or unsafe, based on the extracted rules and their type.

Based on the work reported in [32], in this thesis three probable ACS failure scenarios have been studied. Different types of faults have been injected into the ACS

model and data for nine pre-selected attributes have been collected for faulty, as well as, non-faulty system behavior. These faults are similar to the ones that are often encountered in practice. Next, a neural model was generated training each of the four faulty variables against non-faulty variables and diagnosing faults in the data provided.

Under each failure scenario, a neural model was checked and optimal values were estimated for realizing a non-faulty model. At this stage, we assume that the neural network model is trained sufficiently. Finally, two different rule extraction algorithms have been applied to the neural network model, in order to generate a general rule from the trained neural network. The extracted rules are formally verified using two different formal techniques such as model checker and theorem-prover which was explained in Chapter 4. It has been found that the extracted rules relatively mimic the working of the neural model. The proposed approach manages to prove that there are techniques to formally verify the working of neural networks. Though we have managed to show that neural networks can be used successfully in safety critical systems and the working of the neural network can be understood by the process of rule extraction, there still remain uncertainties with the neural networks. One of these is the training output from the neural network which cannot be completely reliable.

Finally, we point out that limited work has been done within the scope of verification and validation of neural network based safety critical system. This has been the motivation behind our intention to design such a framework.

The contribution of this thesis can be summarized as follows:

(1) Introduced a framework for verification and validation of neural network based safety critical systems.

(2) Developed a neural network model for training and diagnosing faults for a generic attitude control subsystem (ACS) model with faults being injected in the model.

(3) Demonstrated that formal techniques can be successfully used in verification and validation of neural networks used in safety critical applications.

(4) Demonstrated verification and validation of neural networks using the proposed framework under three different ACS failure scenarios thereby illustrating that formally verified neural network based autonomous systems could be used safely in mission critical systems

The thesis can be extended in order to obtain better results. Thus, as part of future research work within our topic, the rule extraction algorithm can be perfected. Furthermore, the same approach can be implemented for dynamic neural networks. At present the neural networks that we have adapted are fixed neural networks, where weights and bias information are extracted from the neural network after they have been fully trained. The complete rule extraction algorithm has been applied using static weights and bias information. The neural network learning process however is continuous and hence dynamic neural networks are better suited for the verification and validation of neural networks used in mission critical applications. Therefore, the proposed framework and the network topology used have to be modified for improved results. Though some of the existing algorithms for rule extraction have provided good fidelity and comprehensibility for neural networks such that they can be used in safety critical applications, a more accurate algorithm is needed. Rule extraction is only one facet of

the verification and validation of neural networks used in safety critical systems. There still remain several barriers to overcome before neural networks can gain full acceptance within the domain of safety critical systems. Formal methods have great potential for proving the correctness of the specification of the model and this thesis has provided an insight into the possible formal techniques that can be employed. This is simply a foundation for the extent of work that can be done with regards to the applicability of formal methods in neural networks.

Bibliography

[1] Tim Menzies, Charles Pecheur "*Verification and Validation and Artificial Intelligence*" Foundations 2002: A V&V Workshop, Maryland, USA, October 2002.

[2] Wei-Tek Tsai, Rama Vishnuvajjala, and Du Zhang "*Verification and Validation of Knowledge-Based Systems*" IEEE Transaction on Knowledge and data Engineering.Vol 11, pp 202-212, January 1999.

[3] Bojan Cukic "*The need for verification and Validation Technique For Adaptive Control System*" Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems, 2001.

[4] Ashish Tiwari, Purnendu Sinha "*Issues in V&V of Autonomous and Adaptive systems*" Proceeding of IEEE CCECE, Montreal, May 2003.

[5] Johann Schumann, Stacy nelson "*Towards V&V of Neural network based controllers*" Proceeding of the first workshop on self-healing systems, Charleston, South Carolina, pp 67-72, 2002.

[6] Brian Taylor, Marjorie Darrah, Christina Moats "*Verification and validation of neural networks: A sampling of research in progress* "In Proceedings of AeroSense. Orlando, FL, pp 21-25, April 2003.

[7] Jason Hull, David Ward, Radoslaw R.Zakrzewski "*Verification and validation of Neural network Accuracy*", Proceedings of the International Joint Conference on Neural Networks, Washington, DC, pp 1657-1662, 2001.

[8] Sampath Yerramalla, Edgar Fuller, Martin Mladenovski and Bojan Cukic"*Lyapunov Analysis for Neural Network Stability in an Adaptive Flight Control System*" Technical Report, Lane Department of Electrical and Computer Engineering, University of West Virginia, WV, USA, July 2003.

[9] Jacek M.Zurada"*Introduction to Artificial Neural Systems*"St.paul,MN,1992

[10] Vincent A. Schmidt and C.L.Philip Chen "*Using the Aggregate Feedforward Neural Networks for Rule extraction*" International Journal of fuzzy systems, Vol.4, No.3, September 2003.

[11] Paulo J.S.Lisboa "*Industrial Use of Safety-related artificial neural networks*" Technical Report, Liverpool John Moores University, School of Computing, Liverpool, 2001.

[12] Vittorio Cortellessa, Bojan Cukic, Diego Del Gobbo, Ali Milli, Marcello Napolitano" *Certifying Adaptive Flight Control Software*" Technical Report, Department of Electrical and Computer Engineering, University of West Virginia, July 2003.

[13] Neil Storey," *Safety Critical Computer Systems*". Harlow: Addison-Wesley, 1996.

[14] Wlodzislaw Duch,Rafal Adamczak,Krzysztof Grabczewski "*Extraction of logical rules from training data using backpropagation networks*" Technical Report, Nicholas Copernicus University, Grudziadzka 5, pp 87-100 Torun, Poland, 1996.

[15] Vasile Palade, Daniel Neagu, Ron.J.Patton *"Interpretation of trained neural network by rule extraction"* Lecture Notes in Computer Science-Springer Verlag, LNCS2206-Computional Intelligence: Theory and Application, pp 152-161,September 2001.

[16] Robert Andrews, Joachim Diederich, Alan B.Tickle "*A Survey And Critique of Technique For Extracting Rules From Trained Artificial Neural Networks*" Knowledge Based Systems 8, pp 373-389, 1995.

[17] Fu, LiMin " *Rule generation from neural networks* " IEEE transactions on systems, Man, and cybernatics, vol 28, no 8 pp 1114-1124, 1994.

[18] Sebastian Thrun "*Extracting Provable Correct Rules from Artificial Neural Networks*" Technical Report, University of Bonn, 1993.

[19] Ashish Darbari "*Rule Extraction from Trained ANN: A survey* Technical Report WV-2000-03, Knowledge Representation and Reasoning Group, Department of Computer Science, Dresden University of Technology, Dresden, Germany, 2000.

[20] Jia-Zhou He, Zhi-Hua Zhou, Xu-Ri Yin, Shi-Fu Chen "*Using Neural Networks for fault diagnosis*" In proceedings of the IEEE-INNS-ENNS International joint Conference on Neural Networks, Como, Italy 2000.

[21] Chrissanthi Angeli, Anna Chatzinikolaou "*On-line Fault Detection Techniques for Technical Systems: A survey"* International Journal of Computer Science &Applications Vol I, No.1, pp.12-30, 2004.

[22] Vincenzo Piuri "*Continuous Learning:A Design Methodology for Fault-Tolerant Neural Networks* " Proceedings of the third international conference on industrial and engineering applications of artificial intelligence and expert systems-Volume2, Charleston, South Carolina, United States, pp 1019-1029, 1990.

[23] Saroj. K. Das, John Fox, David Elsdon, and Michelle Hammond, "*A flexible architecture for autonomous agents,*" Journal of Experimental and Theoretical Artificial Intelligence, 1997.

[24] Guido Bologna "*A Study on Rule Extraction from Neural Networks Applied to Medical Database*" Technical Report, National University of Singapore, School of Computing, Lower Kent Ridge road, 119260 Singapore.

[25] Gary Montague, Nicholas .J. Morris, and Peter Turner, "*Process Systems Applications of Artificial Neural Networks*,"presented at Safety-Critical Systems Symposium. 3rd,Brighton, England, 1995

[26] Amitabh Barua "A Fault-Tree Approach for Identifying Causes of Actuator Failure in Attitude Control Subsystem of Space Vehicles" MASc Thesis, Department of Electrical and Computer Engineering, July 2004.

[27] JudeW.Shavlik, "*A Framework for combining symbolic and Neural Learning*" Technical Report, Computer Science Department, University of Wisconsin, Madison, 1992.

[28] Ismail A.Taha and Joydeep Ghosh "*Symbolic Interpretation of Artificial Neural Networks*" IEEE Transactions on Knowledge and Data Engineering, Vol 11, No.3,May/June 1999.

[28] Mark W.Craven, Jude W.Shavlik "*Using Sampling and Queries to Extract Rules from Trained Neural Networks*" Appears in Machine Learning: Proceedings of the Eleventh International Conference, San Francisco, CA, 1994.

[29] Kim G. Larsen, Paul Petterson and Wang Yi " *UPPAAL in a Nutshell*" In Springer International Journal of Software tools for Technology Transfer 1(1+2),1997.

[30] Sam Owre and Natarajan Shankar "*Writing PVS Proof Strategies*" NASA conference Publication, NASA Langley Research Center, pp 1-15, Sep 2003.

[31] Boser Bernhard, Guyon Isabelle, and Vapnik Vladimir " A training algorithm for optimal margin classifiers" In Proceedings of the Fifth Annual Workshop on computational Learning Theory, 1992.

[32] Johann Schumann, Pramod Gupta and Stacy Nelson "*On Verification and Validation of Neural Network Based Controllers*" In the Proceedings of Engineering Applications of Neural Networks 2003.