Fault Diagnosis of Neural Network Modelled Mechanical Systems using a Sparse Bayesian Learning Framework

Sparsh Sharma

Master of Science Thesis

Fault Diagnosis of Neural Network Modelled Mechanical Systems using a Sparse Bayesian Learning Framework

by

Sparsh Sharma

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday May 28, 2019 at 9:00 AM.

Student number:	4618564	
Project duration:	July 7, 2018 – May 28, 2019	
Thesis committee:	Prof.dr.ir. J. Hellendoorn,	TU Delft, supervisor
	Prof.dr.ir. Martijn Wisse,	TU Delft
	Dr.ir. Riender Happee,	TU Delft

TU Delft

Abstract

The increasing complexity of mechanical systems has resulted in an increased usage and dependence on data driven modelling techniques in order to obtain simple yet accurate models of these systems. Neural networks have emerged as a popular modelling choice due to their proven ability to learn complex nonlinear relationships between inputs and outputs of any given system. Moreover, they are capable of generalizing on data that they have not been trained on. The downside of modelling with neural networks is that they do not provide any insight into the dynamics of the system they model. This limits the application of neural networks in carrying out fault diagnosis of mechanical systems to just the fault detection and isolation (FDI) tasks. While in some applications this may be sufficient, sometimes alongside FDI, it is also desirable to carry out a fault identification task in order to determine the necessary adjustments to bring the faulty system back to its normal operating condition. This thesis explores the possibility of carrying out a fault identification task alongside an FDI task for a mechanical system that has been modelled by a neural network.

Traditionally, the weights of a trained neural network represent the strength of a connection between the two neurons they connect. The possibility of an existing correlation between the weights of a neural network and the properties of the mechanical system being modelled is a concept that has not been fully explored yet. This study considers that such a relation exists, implying that the change in certain properties of the mechanical system due to the occurrence of a fault can be related to a change in the corresponding weights of the neural network modelling the system. Consequently, the change in weights of the neural network could give an insight into the fault occurrence in a mechanical system. Taking this idea forward, two fault diagnosis algorithms have been proposed in this study - a fault detection algorithm using adaptive threshold, and a fault isolation & identification algorithm based on sparse Bayesian learning framework. The proposed algorithms were tested on (hypothetical) faulty linear and non-linear systems. The results show that the adaptive threshold based fault detection algorithm was successful in detecting the occurrence of faults in the linear system. For the non-linear system, although a simplistic neural network was used to model the system, the fault detection algorithm was still successful while returning few and sparse false positives and negatives. The fault isolation & identification algorithm was also successful in isolating and identifying all the changed weights in the neural networks modelling the system for both linear and non-linear cases. Although the algorithms proposed show promising results for the experiments conducted, further research is needed to establish the suitability of using them in real world applications.

Acknowledgement

This master thesis would not have materialized without the help and support of many individuals whom I wish to acknowledge. First and foremost, I would like to express my sincere gratitude to my supervisor, Prof.dr.ir. J. Hellendoorn, without whom this thesis would not have been completed. I would like to thank him for stepping in and providing guidance and support when it was needed the most.
I would like to thank dr. Wei Pan for introducing me to the research idea explored in this thesis, as well as for his guidance during the initial phase of this research. I am also indebted to Hongpeng Zhou for his technical advice, patience as well as words of encouragement over multiple conversations we had.
My experience in Delft, for the most part, has been shaped by my friends and fellow students. I would like to thank Janani Venkatasubramanian, Aishwarya Karthikeyan, Shambhuraj Sawant and Apourva Parthasarathy for being amazing friends with whom I have shared moments of deep apprehension as well as exhilaration and who have made Delft feel like home. I am grateful to Malvika Dixit for her valuable support and motivation. Thanks are also due to my flatmates Ajith Anil Meera, Pratik Mishra, Mihir Tandon, Ajinkya Krishnakumar, Diwakar Harsh and Tapasya Mukkamala for the wonderful conversations, lunches, dinners and game nights that made even the most stressful times better.
Finally, but most importantly, I would like to thank my family, and specially my mom, dad and my sister for their unconditional love and support through all the tough times. They have stood by me in light and dark and have never failed to inspire me to better myself, personally and professionally. They will always be my greatest strength, and to them I dedicate this work.

Delft University of Technology — Sparsh Sharma
May 28, 2019

Contents

List of Figures **ix**

List of Tables **xiii**

1 Introduction **1**
1.1 Introduction . . . 1
1.2 Related Work . . . 2
1.3 Scope of thesis . . . 6
1.3.1 Problem Description . . . 6
1.3.2 Research Question and Objectives . . . 6
1.4 Thesis Outline . . . 7

2 Technical Preliminaries **9**
2.1 Modelling of systems . . . 9
2.2 Fault Diagnosis of Mechanical Systems . . . 14
2.2.1 Fault Detection . . . 14
2.2.2 Fault Isolation . . . 18
2.2.3 Fault Identification . . . 19
2.3 Sparse Signal Recovery Problem . . . 22
2.3.1 Sparse regression techniques . . . 24
2.4 Discussion . . . 30

3 Fault Detection, Isolation and Identification **31**
3.1 Fault Detection Algorithm . . . 31
3.1.1 Adaptive Threshold . . . 31
3.2 Fault Isolation and Identification Algorithm . . . 33
3.3 Discussion . . . 51

4 Experiments **53**
4.1 Experiment 1: Linear system modelled by neural network without hidden layer . . . 53
4.2 Experiment 2: Linear system modelled by neural network with a hidden layer . . . 57
4.3 Experiment 3: Nonlinear system modelled by neural network with a hidden layer . . . 64
4.4 Discussion . . . 70

5 Conclusions and Future Work **71**
5.1 Conclusions . . . 71
5.2 Recommendations for Future Work . . . 72

Bibliography **77**

List of Figures

1.1 Broad classification of the different analytical redundancy fault detection and isolation methods [1] and the focus of this research (highlighted) 3

2.1 Classification of the different modelling approaches. 9
2.2 Typical architecture of a neural network showing its layers, neurons and weights [2]. 11
2.3 Forward propagation process for neural network training [3]. 12
2.4 Backward propagation process for neural network training [4]. 13
2.5 Neural network modelling error between true output (blue) and predicted output from model (orange) [5]. 13
2.6 Fault occurrence as a single event in the time-series data (Left); as an ECG signal and a sub sequence (portion in red) in the time-series (Middle) ; and as the entire time-series (Right). [6] 15
2.7 The SVM its working 17
2.8 The transformation of data from input space to feature space using the Kernel function $\phi(x)$[7]. 18
2.9 The inference network showing a mapping between the faults and symptoms for these faults for an oil refinery [8]. 19
2.10 Overview of the working principle and mathematics of a state-observer/Kalman Filter [9]. 20
2.11 Typical structure of a sparse signal recovery problem [10]. 22
2.12 EEG/MEG setup for inferring brain activity of different parts of the brain in response to stimulus [10]. 23
2.13 Overview of the classification of sparse recovery algorithms. 24
2.14 Greedy sparse recovery algorithm [10]. 24
2.15 Graphical representation of the L1 norm (left) and the L2 norm (right). 26
2.16 MAP estimation framework to identify sparse vector x from known measurements y [10]. 27
2.17 Typical Sparse Bayesian Learning framework with a list of hyper-parameters γ, corresponding elements in sparse vector x and available measurements y [10]. 28

3.1 Fault detection algorithm. 32
3.2 Comparison between a static and an adaptive/dynamic threshold for the purpose of a fault detection based on the residual generation. 33
3.3 Neural network architecture without a hidden layer used for modelling the linear system. 34
3.4 Reconstruction of neural network weights by reformulating into a sparse signal recovery problem. 35
3.5 Neural network architecture with a hidden layer. 37
3.6 New neural networks (right) trained on the prediction errors (left) at the output states. 39
3.7 (Left) Two neural networks trained on the prediction errors of the output states; (Right) The likely location of the changed weights narrowed to the two dotted weights in red. 41
3.8 Likely location of the changed weights narrowed down to the four dotted weights in red and blue. 41
3.9 Calculating the values of the neurons 'a' and 'b' in the hidden layer at time step t = 5. 42
3.10 Calculating the value of the three unknown hidden layer neurons a, b and c. . 42

3.11 (Left) Prediction errors at output states y_1 and y_2. (Middle) Neural network trained on prediction error at output state y_1. (Right) Likely weights that changed in the neural network after the occurrence of fault. . . . 43
3.12 (Left) Prediction errors at output states y_1 and y_2. (Middle) Neural network trained on prediction error of output state y_1 and y_2. (Right) Likely weights that changed in the neural network after the occurrence of fault. . . . 44
3.13 Case 3: Architecture of the neural networks (right) that are trained on the prediction errors at the output states (left). . . . 45
3.14 (Left) Prediction error in the output state y_1. (Middle) Neural network trained on the prediction error. (Right) Red dotted weights represent the likely weights that changed after fault. . . . 46
3.15 (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Red dotted weights represent the likely weights that changed after fault. . . . 48
3.16 (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Red dotted weights represent the likely weights that changed in the neural network after fault. . . . 49
3.17 (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Red and blue dotted weights represent the likely weights that changed in the neural network after fault. . . . 50
3.18 (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Blue dotted weights represent the likely weights that changed in the neural network after fault. . . . 51

4.1 Experiment 1: Training data set corresponding to the linear system's six inputs (left) and two output states (right) under normal operating conditions. . . . 54
4.2 Experiment 1: Cross-validation of the trained neural on training (left) and test data sets (right). . . . 54
4.3 Experiment 1: Effect of the fault on the linear system's output states y_1 (top) and y_2 (bottom). . . . 55
4.4 Experiment 1: Fault detection algorithm monitoring for faults at the output states y_1 (top) and y_2 (bottom) of the system. . . . 56
4.5 Experiment 1: Classification made by the fault detection algorithm for each time-step for output states y_1 (left) and y_2 (right). . . . 57
4.6 Experiment 2: Training data set corresponding to the linear system's six inputs (left) and two output states (right) under normal operating conditions. . . . 58
4.7 Experiment 2: Cross-validation of the trained neural on training (left) and test data sets (right). . . . 58
4.8 Experiment 2: Effect of the fault on the linear system's output states y_1 (top) and y_2 (bottom). . . . 60
4.9 Experiment 2: Fault detection algorithm monitoring for faults at the output states y_1 (top) and y_2 (bottom) of the system. . . . 61
4.10 Experiment 2: Classification made by the fault detection algorithm for each time-step for output states y_1 (left) and y_2 (right). . . . 61
4.11 Experiment 2: Prediction errors on the output states y_1 (top) and y_2 (bottom) of the system. . . . 62
4.12 Experiment 2: Neural networks trained on the prediction errors on output states y_1 (left) and y_2 (right). . . . 62
4.13 Experiment 3: Training data set corresponding to the linear system's six inputs (left) and two output states (right) under normal operating conditions. . . . 64
4.14 Experiment 3: Cross-validation of the trained neural on the training (left) and test (right) data sets. . . . 65
4.15 Experiment 3: Trained adaptive threshold models for output states y_1 (left) and y_2 (right). . . . 65
4.16 Experiment 3: Effect of the fault on the nonlinear system's output states y_1 (top) and y_2 (bottom). . . . 66

4.17 Experiment 3: Fault detection algorithm monitoring for faults at the output states y_1 (left) and y_2 (right) of the system. 67
4.18 Experiment 3: Classification made by the fault detection algorithm for each time-step for output states y_1 (top) and y_2 (bottom). 68
4.19 Experiment 3: Prediction error on the output states y_1 and y_2 of the system. Part in red is the prediction error after the occurrence of the fault from time-step 401 onward 68

5.1 Identifying the equation of the system from the available measurement data using basis functions. 75

List of Tables

3.1 Linear systems: Distinctive patterns in the values of the weights of the new neural networks trained on the prediction errors of system's output states y_1 and y_2. 40
3.2 Nonlinear systems: Distinctive patterns in the values of the weights of the new neural networks trained on the prediction errors of system's output states y_1 and y_2. 47

4.1 Experiment 1: Mean-squared error on the training and test data sets. 54
4.2 Experiment 1: Classification results of the fault detection algorithm. 56
4.3 Reconstructed weights for experiment 1. 57
4.4 Experiment 2: Mean-squared error on the training and test data sets. 59
4.5 Experiment 2: Classification results of the fault detection algorithm. 60
4.6 Experiment 2: Value of the weights of the two new neural networks trained on the prediction errors on the output states of the system. 63
4.7 Reconstructed weights for experiment 2. 63
4.8 Experiment 3: Mean-squared error on the training and test data sets. 65
4.9 Experiment 3: Classification results of the fault detection algorithm. 67
4.10 Experiment 3: Identified pattern in the value weights of the two new neural networks that are trained on the prediction errors on the output states (y_1 and y_2) of the system. 69
4.11 Reconstructed weights for experiment 3. 70

5.1 Results obtained using the SBL algorithm for the reconstruction of red and blue weights lying in the W_1 and W_2 layer. 74

1 Introduction

1.1. Introduction

The last few decades have seen an increased usage and dependence on mathematical models across several engineering disciplines such as the aerospace, automotive and process engineering industries where these models are used for the purpose of designing, optimizing, testing, controlling and maintaining of engineering systems. Even in applications outside of engineering such as health care, there has been an increased interest in model-based techniques following a gradual ongoing shift towards personalized health care. Over the years, the increasing demands on the performance, efficiency, safety and environmental aspects have pushed engineering systems to become increasingly complex in both their operating dynamics and their interactions with other sub-systems that surround them (in case of distributed architectures).The downside of this increasing complexity is that it poses a major challenge in accurately modelling such systems. This is because there is a high chance that the engineering insights and governing equations describing the working dynamics of the complex system and its subsystems may be unknown, or too difficult to analytically express. Even when equations are known, the mathematical model may be sensitive to modelling errors which in turn affects the success of the model based method. Lastly, if known equations are overly complex in their analytical expression, they might be computationally expensive to solve in real time and therefore unsuitable for application. As a result, there has been an increased usage of data-driven modelling techniques such as black-box models, which can obtain accurate models directly from the available measurement data from the system, without requiring any/much knowledge of the dynamics behind it. However, such models provide limited insights in the dynamics of the modelled systems. Neural networks are one of the most commonly used black-box modelling techniques.

One common requirement for the systems employed across the engineering and non engineering disciplines mentioned earlier is their (system's) reliable operation in an uncertain and (temporally and spatially) changing environment. Consider for example a mechanical system employed in a process or manufacturing engineering application. There are time instances during the system's running operation when a fault(s) occurs, especially when the system is operated at its peak load/capacity conditions for extended periods in order to meet production demands. After the occurrence of this fault, certain properties of the system change as a result of which the system dynamics are affected. This in-turn means that the identified black-box model of the system is no longer able to describe the dynamics corresponding to the current faulty system. This renders the model useless for the purpose of control applications of the system unless the occurrence of the fault in the system can be detected at the earliest and a diagnosis of the system can be carried out in order to identify the cause of this fault and the parameters of the system that have changed as result of this fault. Performing these tasks are collectively referred to as a fault diagnosis of the faulty system and in such high-cost safety critical systems it is advantageous to have a fault diagnosis system in place. The aim of this research is to propose fault diagnosis algorithms that can carry out the fault diagnosis of faulty systems that have been modelled using neural networks.

1.2. Related Work

Fault diagnosis techniques can be roughly classified under two main categories namely: physical redundancy and analytical redundancy. In the physical redundancy approach, redundant sensors are installed on board the operating system. During the system's normal operation, only one sensor is required for the purpose of measuring or controlling a particular variable. However, following the occurrence of a fault, the redundant sensors provide reliable measurements which allows for carrying out a fault diagnosis and maintaining control of the system. Despite the advantage of being easy to implement, some of the downsides of this approach include high maintenance and operational costs, and an increase in the overall size, bulk and complexity of the system which is undesirable for already complex systems. Moreover, this approach is not suitable for systems where the parameters of the system can not be directly measured by sensors. On the other hand, the analytical redundancy approach involves constructing a mathematical model representing the dynamics of the system. The mathematical model is then compared with the actual system during its operation in order to detect and isolate any occurrence of fault in the system. The figure 1.1 shows a broad classification of the various analytical redundancy based approaches.
The fault detection and identification (FDI) methods shown in this figure can be broadly classified as model-based and knowledge-based methods. Model-based quantitative methods require a prior knowledge of the governing equations of the system in order to construct a mathematical model that represents the dynamics of the system. While adopting this approach is likely to result in an accurate mathematical model of the system, the increasing complexity of systems means that often the exact governing equations of the system are either not known or partially known. The alternative of making some assumptions on the system in order to obtain a simplified model results in an inaccuracy between the modelled and real system, which could in-turn affect the fault detection accuracy. Conversely, knowledge-based methods do not require prior knowledge of the governing equations of the system. For example, consider quantitative knowledge-based methods that make use of the abundantly available measurement data from the system in order to construct a mathematical model of the system. This type of modelling approach is also commonly referred to as a data-driven modelling approach for which black-box modelling methods such as neural networks and statistical methods are widely used. Although an accurate model is obtained, some disadvantages of quantitative knowledge-based methods are that they require lot of data from the system in order to construct a model. Moreover, the model obtained does not provide much insights into the dynamics of the system itself. On the other hand, qualitative knowledge-based methods construct models based on the vast available knowledge of the system's working dynamics in its normal and faulty operational modes. The models are constructed with the aim of emulating the decision-making ability of a human expert. FDI using these models is carried out based on logical reasoning based on an inbuilt inference network or rule-based network that links certain observed phenomena/symptoms of the system to its current health condition and then suggests the necessary actions to be carried out, similar to how a human expert would. The major disadvantage of the qualitative knowledge-based methods is the problem of knowledge-acquisition. Although quantitative knowledge-based methods have some drawbacks, their ability to construct an accurate model of the system purely on measurement data have made them an increasingly popular and suitable option among researchers in recent times.

Fault detection and isolation methods

- **Statistical based FDI methods**
  Achmad et. al. [11] used Support Vector Machine (SVM) classifiers for FDI of an induction motor having several fault modes. Prior to the classification process, raw data was subjected to Principal Component Analysis (PCA) and Independent Component Analysis (ICA) data dimensionality reduction methods to improve the SVM's classification performance. The SVM's fault classification accuracy on test data was compared for four

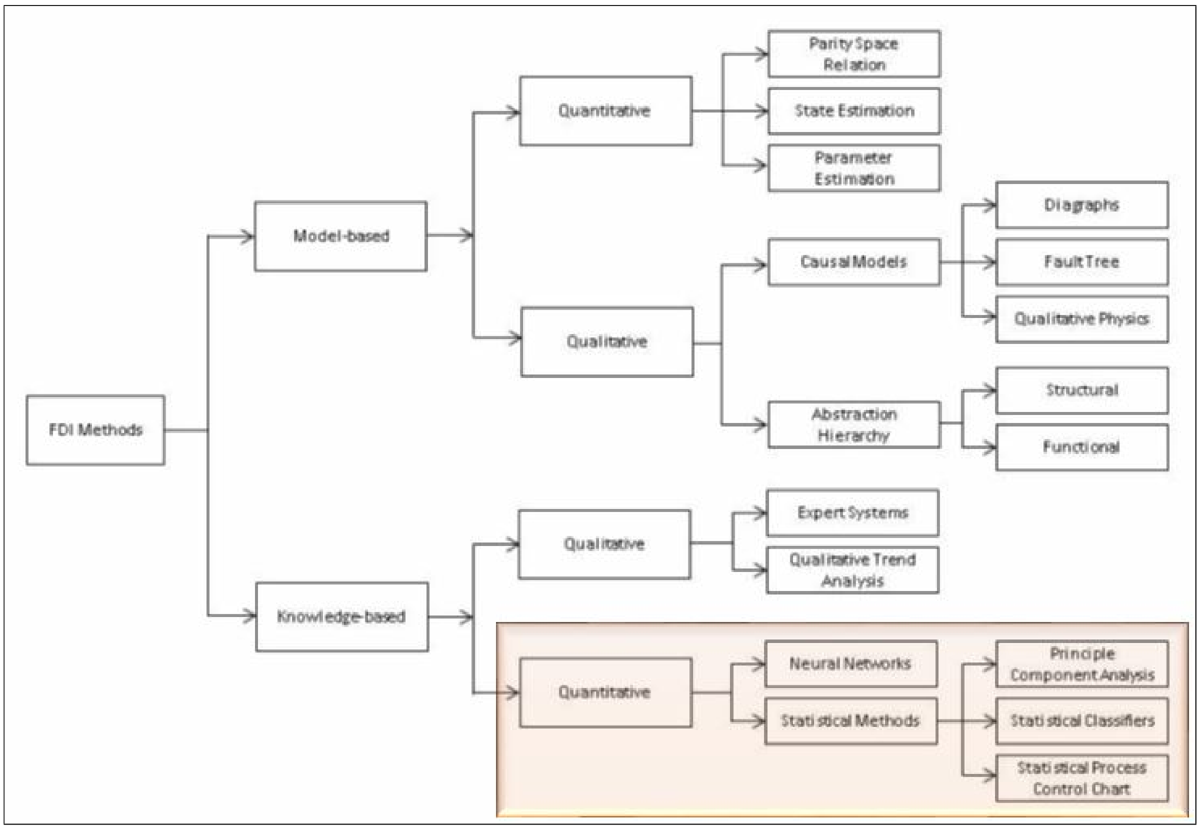


Figure 1.1: Broad classification of the different analytical redundancy fault detection and isolation methods [1] and the focus of this research (highlighted)

separate cases: data dimensionality reduction of the test data carried out by linear and nonlinear variants of PCA and ICA. It was found that kernel based methods resulted in a more accurate classification when compared to their linear counterparts. Moreover, kernel ICA method was shown to outperform kernel PCA method.
SVM based FDI approach has also been used in several other applications such as FDI of gears [7], wind turbines [12] and turbo-pump rotors [13] to name a few.
Setu et. al. [14] performed a fault diagnosis of HVAC chillers that had 27 modes of normal operation and 8 faulty conditions with each faulty condition having 4 severity levels of the fault. A Partial Least Squares (PLS) approach was used to obtain a nominal model of the system. A fault in the system was detected by performing the generalized likelihood ratio test (GLRT) on residuals that were generated by comparing outputs of the nominal model with actual measurements. Fault isolation was carried out by 3 different types of classifiers namely SVM, PLS and PCA. A comparison of the classification accuracy for all three methods in each of the 27 operating modes showed that the classifiers outperformed one another for different operating conditions. Each classifier was found to have a classification accuracy of at least 95 %, with SVM performing best on average. The fault identification task to identify the severity of the fault was performed by another PLS model that was trained separately on the four severity levels of each fault type.
PLS based FDI approach has also been used in several other applications such as the fault detection and isolation of robotic manipulators[15] and chemical systems [16] to name a few.

PCA has been used in areas such as data dimensionality reduction, pattern recognition and feature extraction. Its simplicity and efficiency in handling large volumes of data has lead to its usage in process monitoring and FDI. Lee et. al. [17] proposed a nonlinear kernel principal component analysis (KPCA) approach and compared it to linear PCA for carrying out a fault detection of a biological waste-water treatment process. The nonlinear kernel function was used to project the data corresponding to the normal operating condition of the system to a feature space from which principal components were extracted. This was used to calculate the T^2 and squared prediction error (SPE) monitoring statistics of the normal operating data, from which the control limits of the T^2 and SPE charts were set. The T^2 statistic provides a measure of the variation within the KPCA model. The goodness of fit of a sample to the KPCA model is given by the squared prediction error (SPE) statistic. A fault in the system is detected when the monitored T^2 and SPE statistics cross the pre-defined control limit for these statistics. It was shown that the proposed KPCA method outperformed the linear PCA approach in capturing nonlinear relationships in the process variables, and the fault detection task. The Fisher Discriminant Analysis (FDA) technique does data dimensionality reduction by determining a set of projection vectors that maximize scatter between the classes while minimizing scatter within each class. PCA on the other hand, does this by projecting data in directions of maximum variance without taking into account the information between classes. As a result, in PCA there is a possibility that the identified directions of maximum variance may not necessarily be effective for classification. Therefore in theory FDA is expected to do better because it carries out data reduction while still being able to differentiate among classes. Unlike in PCA, where separate models are identified for the normal operating conditions (fault detection) and for each fault case (fault isolation). In FDA, all data (normal and fault) is used to obtain a single lower dimensional model for carrying out a fault diagnosis. Identical to PCA, in FDA the fault is detected and isolated by using the identified model in combination with the monitored T^2 and SPE statistics. Chiang et. al. [18] compared three techniques namely FDA, PLS and PCA with one another for fault diagnosis of a chemical processes. It was found that FDA and PLS methods were better than PCA at dimensionality reduction from a fault diagnosis point of view. As a result, both FDA and PLS outperformed PCA, with FDA being the best.

The positive results show the suitability of different statistical based modelling techniques for carrying out FDI across various applications. However, a drawback associated with them (except FDA) when it comes to performing a multi-fault classification is that several models have to be trained - one for each fault type. This in turn implies that each of these models have to be checked with in order to make the classification. Moreover, consider systems where multiple faults can be present in the system at a time, the number of models to train based on the different fault combination would raise greatly with a possibility that the classification accuracy decreases. This motivate an investigation into other quantitative knowledge-based approaches such as neural networks for performing FDI of the system.

- **Neural network based FDI**
  Several applications of neural networks have been found in the literature for Fault Detection and Isolation. In one of the earlier works that implemented neural networks for fault diagnosis, Venkatasubramanian et. al. [8] initially identified shortcomings of the Knowledge Based Expert System (KBES) used for FDI of a fluidized catalytic cracking process in an oil refinery. To overcome the shortcomings, a neural network based FDI approach was proposed. The neural network's input and output layers consisted of neurons representing the 18 fault symptoms and 13 possible faults that can occur in the system, respectively. The network was trained on all known 'symptoms-faults' pairs from the inference network in the KBES, and tested for its classification accuracy for single and multiple-faults in the system. The results showed an accurate classification for single faults and a gradual drop in classification accuracy with an increase in the number of faults (multiple-faults case). A possible reason for this could be that the neural network was trained based on the embedded logic in the inference network rather than measurement data corresponding to the different fault cases. In the multiple fault

case, the prediction accuracy increased when an additional hidden layer was added to the neural network architecture. A possible explanation for this is that deeper neural networks (more hidden layers) have a better ability to identify more nonlinear mappings of the system when compared to shallower networks.
Baillie et.al. [19] compared three modeling techniques namely Box-Jenkins linear autoregressive model, back propagation neural networks and radial basis function (RBF) networks for carrying out a fault diagnosis using time-series vibrational signal of a rolling element bearing that has 1 normal and 3 faulty modes. For each technique, a model for each type of bearing fault was identified. A fault in the system was detected by observing the vibration signal of the rotating machine over a fixed time-window and then using the identified models to predict the current vibration level. The model performing the best is said to be indicative of the current operating or faulty mode of the bearing. It was found that the classification accuracy had increased with an increase in the size of the time-window and vice-versa. Results showed that on average neural networks performed better than the other two methods, especially when only shorter lengths of time-series vibrational data was available.
Patton et. al. [20] proposed a neural network based approach for carrying out a fault diagnosis of a nonlinear-laboratory 3 tanks system. The FDI was carried out in two-stages. In the first stage, the predicted output from the trained neural network was compared to the actual system output to generate a residual. In the second stage, the generated residual was passed to the second trained neural network that examined it for the likelihood of a fault(s) in the system. The results showed that the neural network based FDI approach was successful in detecting both single and multiple faults in the system. Moreover, it was shown that small faults could be detected and isolated till a certain limit after which the neural network was no longer capable of being trained on such small faults. A similar approach of carrying out the FDI in two-stages for an electrical power transmission system was seen in the work of Majid et. al. [21].
Chine et. al. [22] used a combined approach of a neural network alongside another algorithm to jointly carry out a fault isolation of a photovoltaic system. This was because the other algorithm was capable of isolating only those fault cases having different combination of attributes and not the fault cases having same combination of attributes. Hence a neural network was trained on these particular fault cases and was shown to be successful in isolating them.
Apart from the literature provided above, neural network based FDI approaches have also been used for several other applications such as FDI of gears [7], gearboxes [23] and induction machines [24] to name a few.

In the literature provided above, the procedure for fault detection remains same: the trained neural network provides an estimated output that is used to generate residuals. These residuals are compared to threshold(s) to detect whether a fault has occurred in the system or not. This threshold could be fixed or adaptive [25]. The fault isolation also remains more or less the same with only a slight variation in its implementation. For example, some researchers have opted to train a single neural network that encompasses fault isolation with fault detection. Other researchers have opted to train a single or multiple neural network(s) for fault isolation. Other subtle differences in the FDI process include selection of number of hidden layers in the neural network. Some researchers have opted for shallow networks while others [22] claim that deep neural networks are capable of performing the FDI better as they can learn the system better. Irrespective of the variations in the approaches taken, it can be concluded that neural networks are suitable candidates for performing or assisting in the FDI over a large and diverse range of systems.

Fault identification methods
The quantitative knowledge based approaches discussed above are limited to detecting and classifying the type of fault that has occurred in the system. While in various applications this is sufficient, sometimes alongside the FDI, it is also desirable to determine the necessary

adjustments in the parameters to bring the faulty system back to its normal operating condition. This requires a determination of the size and time-variant behaviour of the occurred fault. This is referred to as fault identification. Within existing literature, fault identification has been shown to be successfully carried out by: physical redundancy approaches, state observers/estimators [26], Kalman Filters (KF) [27] and reconstruction based contribution approaches using statistical models like PCA [28] and PLS [29]. Literature on using neural networks for carrying out a fault identification for sensor and actuator faults was found in applications such as flight control systems [30] and UAVs [31]. However, to the author's best knowledge, none of the existing literature has looked at using neural networks to carry out a fault identification on process faults for estimating the changed parameters of the faulty system. This research investigates the possibility of doing so.

1.3. Scope of thesis

1.3.1. Problem Description

Consider any complex mechanical system in a process or manufacturing engineering application such as a boiler, mill, etc. In the absence of governing equations or any engineering insight into the mechanical system, the system is chosen to be modelled by a neural network from available time-series measurement data that is obtained from the system while it is operating in its normal (non-fault) state. Traditionally, the weights of a trained neural network represent the strength of a connection between the two neurons it connects. However, the possibility of a correlation between the weights of a neural network and the properties of the mechanical system being modelled is a concept that has not been fully explored yet. This thesis follows the idea that such a correlation exists. Now consider that the mechanical system (along with the trained neural network of the system in parallel) starts its operation at time step 0 and continues to operate under in its normal (non-fault) state till time-step 't'. During its operation, suddenly at time-step 't+1' a fault occurs in the system as a result of which certain properties of the system change. These changes can be related to a change in the values of certain weights in the neural network that correlate to the changed properties. After the occurrence of this fault, from time-step 't+1' onward, the trained neural network model is unable to provide a reliable estimate of the system's output states and therefore this model can no longer be used for the purpose of providing any aid for controlling the system or any other applications unless a fault diagnosis of the faulty system is carried out.

1.3.2. Research Question and Objectives

The research question can be stated as follows:
How can a fault diagnosis (fault detection, isolation and identification) task be carried out for a mechanical system modelled by a neural network?

Based on the stated research question, the aim of this research is to propose suitable fault diagnosis algorithms that are capable of carrying out a fault diagnosis of a neural network modelled mechanical system using available time-series measurement data. Specifically, the objectives of these fault diagnosis algorithms are to:

- Detect that a fault has occurred in the mechanical system and the time-step of its occurrence.

- Locate and identify the weights in the neural network that have changed from their original value after the occurrence of the fault.

- Identify the new values of the weights in the neural network that were found to have changed.

- Test and comment on the suitability of the proposed fault diagnosis algorithms in carrying out a fault diagnosis of any given system modelled by a neural network.

1.4. Thesis Outline

The rest of this thesis is structured as follows: chapter 2 provides a comprehensive technical background on the following three areas: 1) modelling of mechanical systems with a focus on black-box modelling techniques, 2) the sub-tasks that constitute a fault diagnosis process and common approaches for carrying them out, and finally, 3) the sparse signal recovery problem and algorithms that are used to solve such problems with an emphasis on sparse Bayesian learning methods. Chapter 3 proposes the fault detection, isolation and identification algorithms used to carry out a fault diagnosis task on a faulty system. Chapter 4 is the experimentation and results chapter which tests the suitability of the proposed algorithms for carrying out the fault diagnosis task on different types of systems. Chapter 5 draws some key conclusions based on results obtained. Finally, based on these conclusions, some potential areas of future work are proposed and detailed upon.

2 Technical Preliminaries

This chapter provides a basic yet comprehensive technical overview of the following 3 domains: modelling of systems, fault diagnosis and sparse regression techniques. The aim of this chapter is to establish the required base knowledge of the different concepts belonging to each of these domains which are referred to from time to time in the different sections of chapters ahead.

2.1. Modelling of systems

Modelling techniques can be broadly classified as either data driven or non-data driven depending on whether or not the system model is constructed using the measurement data obtained from the system. In the non-data driven modelling approach, the construction of the system model is completely independent of the measured data, rather the model is constructed completely based on standard engineering principles (Newton's law of motion, conservation of energy and more) and the known insights to the system. Models identified via this approach are more commonly referred to as white-box models. On the other hand, data-driven modelling approaches construct a system model using the measurement data acquired from the system. Grey-box and black-box models are the two types of models that fall under this approach. Black-box models are entirely constructed from the measurement data. Grey-box models can be considered as a hybrid of black-box and white-box models in the sense that their construction requires the combination of both engineering insights of the system and the measurement data. More specifically, certain variables of the grey-box models have to be identified for which the measurement data is used. Therefore the selection of modelling using either a white-box, grey-box or a black-box model depends on the availability of measurement data and physical physical insights of the system. Figure 2.1 shows this classification in the form of a flowchart.

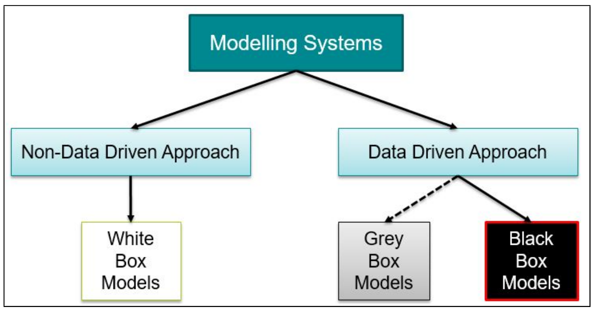


Figure 2.1: Classification of the different modelling approaches.

White-box models have an advantage over grey-box and black-box models in terms of their

transparency, which means that they can provide a good understanding and clear explanation of the inner workings and behavior of the system. This transparency arises from the fact that these models were built using the engineering insights of the system. The disadvantage associated with white-box models is that in the case of complex nonlinear systems (process engineering systems) which may contain many subsystems that interact with one another, it may not be possible to have a complete engineering insight of the system, making the white-box modelling approach too difficult and sometimes impossible. It is also likely that the accuracy of the constructed model might be low. Grey-box models also suffer from this as they too require engineering insights in order to construct the model. Also, they provide a lower transparency as compared to white-box models. The advantage of black-box models is that they are relatively easy to model with as they just require identifying a function (linear or nonlinear) that can fit the observed input to output measurement data from the system, without taking into account the inner working of the system. This makes them a more suitable choice in comparison to white-box models for modelling complex nonlinear systems. The disadvantage of black-box modelling however is that the model identified is unable to provide any transparency, which makes carrying out tasks like maintenance and fault diagnosis of these systems more challenging, but not impossible, as will be explained later in the fault diagnosis section.
Since this thesis is concerned with the fault diagnosis of (mechanical) systems such as those found in process engineering applications, where the system(s) are known to be complex and nonlinear in nature and the engineering insights are unknown, but the measurement data is available, black-box models will be the choice of modelling.
Black-box modelling techniques can be further divided into linear and nonlinear models. Auto-regressive (AR) model, Auto-regressive with Moving Average (ARMA), Auto-regressive with Moving Average with exogenous inputs (ARMAX) models and Output Error (OE) models are examples of a few popular linear black-box models, while Basis Functions and Neural Networks are examples of popular nonlinear black-box models. As mentioned above, the thesis is concerned with modelling of nonlinear systems for this purpose the discussion on linear has been left out but can be referred to here [32].
The discussion on nonlinear black-box models is made bellow:

1. **Basis Functions**
   It is known that a nonlinear function can be represented as a sum of terms belonging to a family of parameterized functions [33]. A popular approach is to search among a set of nonlinear terms belonging to basis functions, for a compact expression that is coherent with the available measurement data. The Taylor polynomial expansion, Volterra expansion and Fourier series are examples of some popular basis functions from which the nonlinear terms can be searched for.
   Success in modelling using basis functions highly depends on selecting a list of appropriate basis functions to search from. This requires having some insight on the system and its domain knowledge in order to make an educated guess on the type of nonlinear terms (polynomial, trigonometric and so one) that are typically expected for modelling such a system. For example, if a mechanical system is known to typically have nonlinear polynomial terms in its expression, then having this insight can help eliminate all basis functions that contain non-polynomial terms in their expansions and help narrow down the search to a limited number of basis functions that do contain these nonlinear polynomial terms of certain polynomial order. If such an insight is missing, then there is an endless number of different types of basis functions and their nonlinear terms to search from, which then becomes a challenging task. This is essentially the downside of modelling using basis functions. The use of basis function for the identification of nonlinear systems can be referred to in [34].

2. **Neural Networks**
   Neural networks (NNs) are one of the most popular modeling techniques for nonlinear systems. They draw their inspirations from the structure and workings of biological neural networks present in human brains. These biological networks are a collection of several interconnected (by synapses) neurons that transmit signals in order to pass

information from one neuron to another. A structure of a typical neural network can be seen in figure 2.2. The neural network contains several interconnected neurons across various layers. The first and last layers are the input and output layers respectively while the layer(s) in between are known as the hidden layer(s). The connection between any two neurons is known as a weight. The number of neurons in the input and output layer is decided by the number of input and output states of the system, with each input/output neuron representing one of the input/output states. The number of hidden layers and neurons in these hidden layers can be considered to be a tuning parameter that is likely dependent on the degree of nonlinearity present in the system. The only way to determine the optimal architecture for modelling the system is by trying with several different neural network architectures to see which one consistently yields a better result.

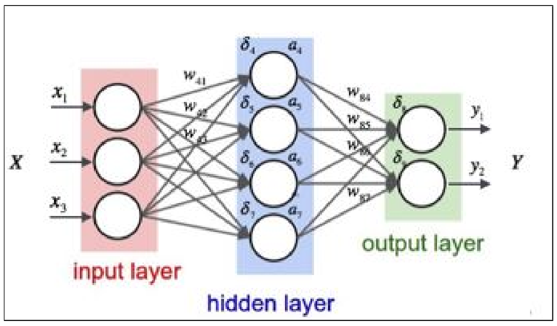


Figure 2.2: Typical architecture of a neural network showing its layers, neurons and weights [2].

Modelling using a neural network basically refers to training the neural network on the available set of measurement data. Training of a neural network is finding the optimal set of values of its parameters (weights) such that the neural network is able to model the system with a high level of accuracy.
Training a neural network is an iterative process consisting of two tasks: Forward propagation followed by backward propagation.

- **Forward propagation**
  The values of the input states are fed to their respective input neurons, and the value of the first neuron in the first hidden layer is calculated by multiplying the value of each neuron in the input layer with their respective weights that connects them to the first neuron in the hidden layer and then summing up all of the products to receive a value. This value is then subjected to the nonlinear activation function (sigmoid, tanh, ReLu, etc) of this hidden layer in order to receive the value of the first neuron in the first hidden layer. This process is then carried out for the remaining neurons in the first hidden layer and then repeated for all the neurons in each hidden layer. The values of neurons in the final hidden layer are then used to calculate the values of the output neurons in the output layer in a similar manner with the exception that the activation function for the final layer is usually a linear one. This process of providing the neural network with a set of input states and calculating the values received at the output neurons is known as Forward Propagation. The forward propagation task in a neural network can be visualized in figure 2.3. Here 'Wij' represents the incoming weights from all neurons 'i' in the previous layer to neuron 'j' in the current layer, 'g' represents the activation function and 'a' represents the neuron value after applying the activation function.

  The final step before carrying out the back propagation is to calculate the error in modelling, this is done with the help of a predefined cost function such as Mean Square Error (MSE), so the MSE between the values received at the output neurons and target (actual output measurement) output for the corresponding input state

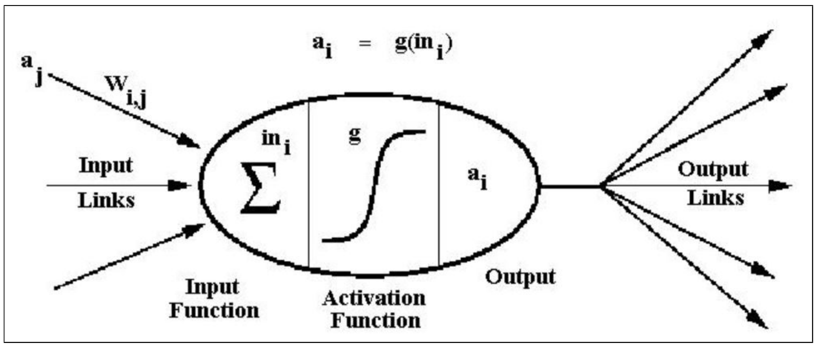


Figure 2.3: Forward propagation process for neural network training [3].

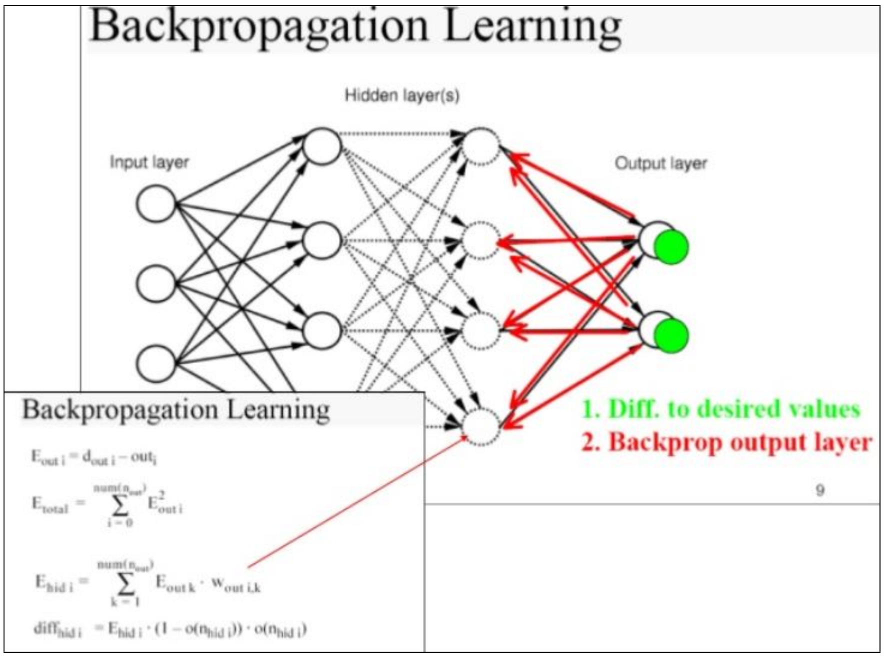
is calculated. This error is then used to carry out the backward propagation task.

- **Backward propagation**
  In the backward propagation step, the values of the neural network parameters (weights) are updated with the intention of reducing the overall modelling error. The magnitude and direction in which the weights of the neural network are to be updated is found out by a process called gradient-descent which requires taking the derivative of the cost function with respect to each weight in the neural network (descent because the goal is to move the parameters in a direction that minimizes the cost function and not maximize it).
  The new set of weights of the neural network are obtained by subtracting the current set of neural network weights by the product of the learning rate with the gradient of the cost function. The learning rate term helps control the magnitude with which the weights are updated in a stable manner. Following this update step, the forward propagation step is repeated for the next input-output data measurement pair with the newly obtained weights. This iterative process continues either for a certain number of iterations or till a stage in the training process is reached when the update terms are negligible. In the later case this implies that the cost function has been minimized and that the neural network has been trained or modelled on the given measurement data. The backward propagation task in a neural network can be visualized in figure 2.4. Here the green circle represents the error between the estimated and target output. The error is then used to calculate new value of weights (indicated in red) by process of backward propagation. The equations represent this process mathematically.

In short, modelling a system with a neural network can be thought of performing an optimization of a cost function which is representative of the fitness of the model output with the actual/target output. The cost is reduced by updating the parameters (weights) of the neural network model in the direction and magnitude as suggested by the gradient of the cost with respect to the neural network model parameters(weights).

Some important points to keep in mind regarding the quality of the identified neural network model (in terms of its fitness) and the interpretation of its weights are:

- **Over-fitting and Under-fitting of model**
  Although the objective is to construct an accurate model of the system with a low modelling error (figure 2.5), this accuracy should not come at the compromise of the generalizing capabilities of a neural network. This means that the neural network model obtained should not have simply memorized (over-fitting) all the inputs and the corresponding outputs that it was trained on, rather it should learn some useful patterns in the data set that allow it to provide a good level of accuracy not only on the data it was trained on, but also for data that it has not yet seen (generalization).
  In order to prevent the neural network from over-fitting, regularization techniques such as dropout [35] and including additional penalty term (L1 and L2 norms) in



$$E_{out\,i} = d_{out\,i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\,i}^2$$

$$E_{hid\,i} = \sum_{k=1}^{num(n_{out})} E_{out\,k} \cdot w_{out\,i,k}$$

$$diff_{hid\,i} = E_{hid\,i} \cdot (1 - o(n_{hid\,i})) \cdot o(n_{hid\,i})$$

Figure 2.4: Backward propagation process for neural network training [4].

the cost function to promote sparsity can be implemented [36]. However, excessive regularization leads to a situation where the neural network model obtained is a poor one which is a sign of an under-fitted model.
The quality of the model in its ability to model and generalize can be tested by dividing the available measurement data set into two data sets: training data set and test data set. The neural network is then trained on the training data set and the obtained neural network model is then validated to see how good its prediction accuracy is on the test data set. If it does well on both data-sets then the neural network can be said to be trained.

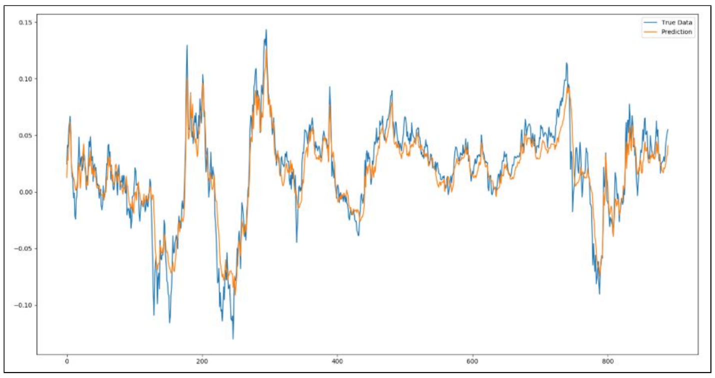


Figure 2.5: Neural network modelling error between true output (blue) and predicted output from model (orange) [5].

- **Interpretation of weights**
  As mentioned earlier, black-box models provide very little insights into the system dynamics of the system being modelled. These insights have to interpreted from the model's parameters. In the case of a neural network model its parameters are its weights. The weights of the neural network are real numbers of certain magnitude. Traditionally these weights provide a simple understanding of how positively (for positive valued weights) or negatively (for negative valued weights) sensitivity two neurons are to data passed between one another. Hence in the case of a trained neural network, the large positive weights imply that the input has a

strong positive proportional correlation to the output states, a large negative weight would imply that the input has a strong inverse correlation of the output states of the system. This is an elementary level interpretation of the weights of a neural network which can be used to gain limited insights into the systems. More work on the interpretation of neural network weights can be found in these papers [37],[38].

Having explained the process of modelling using NNs above, some key advantages associated with NNs that make them an attractive choice for modelling are:

- Superior in their ability to identify and model complex nonlinear relationships between the inputs and outputs of a system.
- Their ability to generalize enables them provide a good estimate on data samples they havn't been trained on. Hence while modelling the system, the neural network is also capable of forecasting which is very useful when it comes to the task of classification as will be explained in the fault detection section later on in this chapter.

Taking into consideration the modelling process and the advantages associated with Neural Networks, they have been chosen as the preferred black-box modelling approach for modelling systems in this thesis.

2.2. Fault Diagnosis of Mechanical Systems

Consider a mechanical system such as those employed in process engineering applications, there are bound to be time instances during its running operation when a fault(s) may occur. The chances of encountering a fault especially increases when the system is operating at its high load conditions for extended periods. In these high-cost safety critical systems it is advantageous to be able to detect the occurrence of a fault at the earliest and carry out a fault diagnosis to identify the cause of this fault and the parameters of the system that have changed as result of this fault.
Before defining what a fault diagnosis is and what its sub-tasks are, it is important to define what a fault is. The IFAC SAFEPROCESS technical committee recognizes the definition of a fault as the definition provided in [39], which states that: "A fault is an unpermitted deviation of at least one characteristic property or parameter of a system from its normal condition." The occurrence of this fault causes a degradation in the system's performance but may not result in complete loss of system functionality.
As per the IFAC SAFEPROCESS technical committee [39], fault diagnosis is a collective of the following 3 tasks sequential tasks:

1. Fault Detection
2. Fault Isolation
3. Fault Identification

The tasks and common methods used to carry them out have been explained in the following sections.

2.2.1. Fault Detection

Fault detection is defined as the determination of the faults present in a system and the time of detection [39]. Fault detection can be thought of as a real-time monitoring and classification process. Carrying out this monitoring process is not so straightforward when it comes to dealing with time-series data. Some of the factors which makes detection in time-series challenging are:

1. There are many ways in which a fault can be defined as shown in figure 2.6. The fault could be a single event or a sub-sequence within the time-series, alternatively it could be the entire time-series itself.

2. Most algorithms used to detect these faults are based on similarity/distance between fault free and faulty systems. Finding a suitable one to apply for a particular type of time-series (periodic, aperiodic, synchronous, asynchronous) is not easy because some of the simpler methods are over sensitive to outliers.

3. Often the length of the training (non-fault) time-series and the test (faulty) time-series are not equal which makes comparing the time-series using the fault-detection algorithms complicated.

4. Sometimes the detection of fault requires the comparison of two or more time-series data which requires them to be in a comparable scale in magnitude.

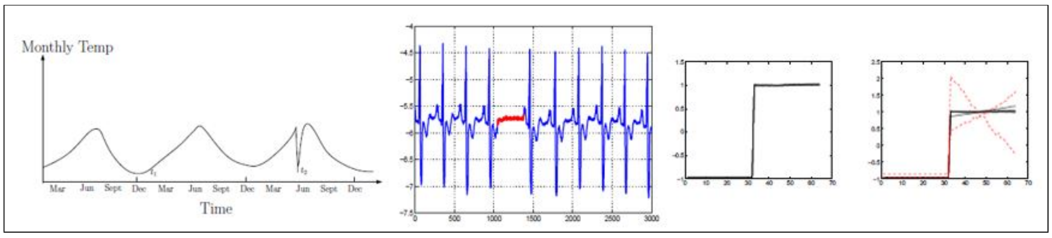


Figure 2.6: Fault occurrence as a single event in the time-series data (Left); as an ECG signal and a sub sequence (portion in red) in the time-series (Middle) ; and as the entire time-series (Right). [6]

To overcome some of the problems stated above, transformation of the data can be done before applying the fault detection algorithms. Some popular transformation techniques include Aggregation, Discretization and Signal Processing. More about these methods can be referred to in [6].
Since the focus of this thesis is in detecting faults in time-series data occurring in mechanical systems, the faults that are of interest to be detected are single-event in nature.
Some of the popular fault-detection algorithms are:

1. **Window-Based** methods divide the entire training and test time-series into windows (sub-sequences) of certain fixed lengths. The score of each test window is calculated by comparing its similarity to the corresponding training window. The scores for each test window is then aggregated to obtain the final score. These application of window based techniques are motivated by the belief that one or more sub sequences could be the cause for the fault.

2. **Similarity-Based** methods calculate the pairwise-proximity between the training and testing time-series data using approximate distance or similarity kernels to compute a score. The scores are calculated by using methods like k-neural network (distance of test time series to its kth nearest neighbour in the training data set) and clustering.

3. **Segmentation-Based** methods partition the training data into a series of segments after which a Finite-State-Automation (FSA) model is learned to model the transition between these series of segments. The idea behind this method is that, a faulty time-series data divided into similar segments will not fit the FSA that was identified for the training time-series data.

4. **Predication-Based** methods identify a function that is capable of modeling the available time-series data from a system. It does so by taking a certain 'n' number of observations of the training time-series data and learns the parameters of the function from this data in order to predict the observations for the subsequent 'm' time-steps. The function identified is said to be trained well if the difference in the predicted output and the true output (prediction error) for these 'm' time-steps is small.
Once the function is identified, it can be said to be represent the system under normal (no-fault) operating conditions. This function, can now be used to forecast the output for any 'n' observations of a test time-series. If the prediction error is large for these 'n'

observations, these data points can be classified as faulty.
Hence, prediction-based fault detection models in essence is a construction of a model from the time-series (measurement) data available which as discussed in the previous section is known as a data-driven modelling approach. This implies that if a black-box modelling technique such as neural networks is chosen for modelling the system, then the same framework can be used be used for the purpose of fault detection.

Window-based, similarity-based and segmentation-based methods are more suitable for their application when faults in the time-series exists as sub-sequences. Prediction based methods on the other hand are more suitable for detecting single event faults in the time-series data. A more detailed overview of the methods mentioned above can be referred to in [6].
The focus of this thesis is on fault diagnosis of mechanical systems, where detecting faults which exist as single events in the time-series data is of interest for which prediction-based fault detection algorithms are a suitable choice. Some examples of prediction based models are ARIMA and ARIMAX models, regression based models such as least squares and lasso regression, Principal Component Analysis (PCA) and state-observers (Kalman filters). Considering that fault detection is essentially monitoring and classifying each data point as a ’fault’ or ’not fault’, it should come as no surprise that, besides the methods mentioned above, classifiers such as Support Vector Machines (SVMs) and Neural Networks are ideal candidates for fault detection.

Some selected popular fault detection methods found in literature are described bellow:

1. **Principal Component Analysis**
Principal Component Analysis (PCA) is a statistical approach that applies an orthogonal transformation to identify orthogonal vectors with the intention of converting the given set of measurement data containing possible correlations among its variables/features into a set of linearly uncorrelated variables known as the principle components. The identified orthogonal vectors otherwise known as Principle Components (PC) are in the direction where most of the variation in the measurement data can be captured. The PCs identified are in their decreasing order of variance that they capture in the data set. In short, PCA can be thought of as a data dimension reduction technique where by selecting just the first few PCs, enough variance in data has been captured as a result of which the rest of the variables/features (PCs) can be neglected or discarded as they don’t play as much role in explaining the correlation in data. For the reduced data set obtained, it is much easier to apply some statistical tests to differentiate the data points that are correlated and uncorrelated to one another.
Mathematically, for any normalized measurement data matrix ‘X’ (where the rows represent the time samples and the columns represent the variables of the system) obtained from a system, the data matrix can be expressed as sum of two terms $\hat{X}$ and E [40] as shown in equations 2.1 and 2.2.

$$X = \hat{X} + E \tag{2.1}$$

$$\begin{aligned} \hat{X} &= TP^T = \sum_{i=1}^{l} t_i p_i^T \\ E &= T_e P_e^T = \sum_{i=l+1}^{m} t_i p_i^T \end{aligned} \tag{2.2}$$

Where, $\hat{X}$ is the predicted portion that lies in the principal component subspace and E contains the residuals that lies in the residual subspace, i.e., E is the the data ”not-explained” by the PCA model. The matrices T and P are the score and loading matrices, respectively.

Applying PCA for fault detection:
Due to the sensor correlation in physical processes, only a few principal components are needed to capture the data variance and help in fault detection and identification. The faults can be detected by monitoring and applying statistical tests to the generated residuals in residual space (E) in order to detect faults. The Squared Prediction Error (SPE) approach used by Wise and Ricker [41] is one such approach where the Euclidean norm of the residuals in the different rows of E are measured and then used to detect the fault. Another approach is the Sensor Validity Index (SVI) proposed by Dunia et al [28] where the ratio of residuals is taken in order to obtain a SVI which lies between 0 and 1 where values close to 0 indicate a fault and values near to 1 indicate no fault. H. Henry Yue et al. in their work [42] combine the SPE and T^2 index into a single index for fault detection. S.Joe Qin in his paper [43] provides an overview of these and more statistical metrics.

2. **Support Vector Machines**
Support Vector Machines (SVM) is a classifier proposed by Vapnik [44] based on statistical learning theory. The most basic version of SVM performs a binary classification, that is either the data falls in class A or class B, in the case of fault detection this can be thought as class 'fault' and class 'not-fault'. The SVM and its working can be visualized in figure 2.7. The SVM performs this classification by identifying a linear boundary (solid line in figure) which can be placed between the two classes (squares and circles in the figure). The orientation of this boundary (w) is decided by trying to maximize its (boundary) distance to the nearest data points falling on either side of it. The distance being maximized is otherwise known as the margins (dotted lines). The data points belonging to either class which was used to identify the margins are known as the support vectors (line going through grey square and circle in figure). Once the margin is identified, the boundary is placed along the middle of the margin. The identified support vectors and boundary equations are enough to carry out the classification task.

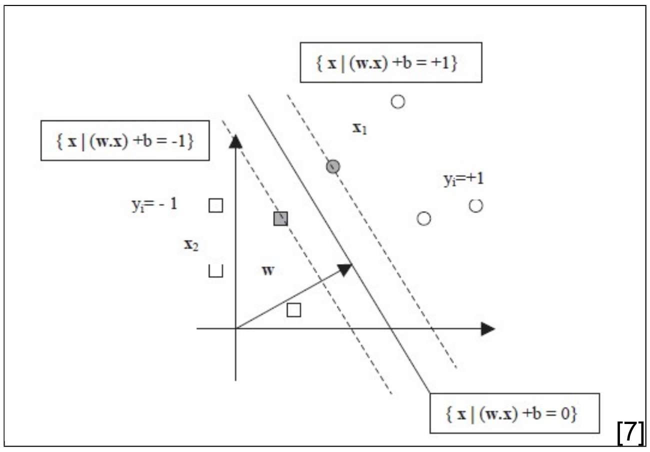


Figure 2.7: The SVM its working

This basic form of SVM places a linear boundary between the classes which it assumes are linearly separable but in the case that the classes are not linearly separable, the original data can be projected to a different feature space with the help of Kernel functions. The result of this projection is that the data points in this feature space are now linearly separable and the SVM can now be applied for the classification task. This procedure of projecting the data to a different feature space using Kernal functions is known as transformation and can be visualized in figure 2.8.

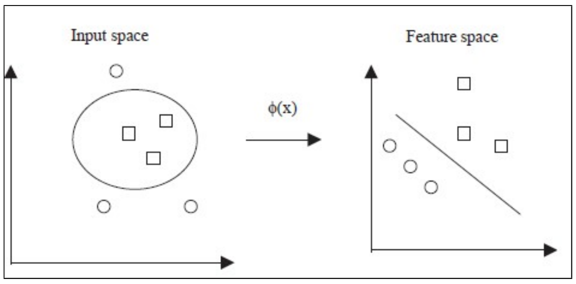


Figure 2.8: The transformation of data from input space to feature space using the Kernel function $\phi(x)$[7].

SVM for fault detection: For the fault detection task, the SVM can be modelled using the available time-series measurement data, where it is already known which data points correspond to the system's normal and faulty operations. Following the training of the SVM on the measurement data, the equations for the linear boundary/hyperplane and support vectors for both classes (fault and not-fault) have been identified. The classification of any new time-series data point into either the faulty or normal system operation classes can be done by substituting this data point into the equations identified. The work by Samanta et al. in [7] and [45] show a comparative study between SVM and neural networks for the task of fault detection in gears and roller bearings respectively.

Finally it is important to note that SVMs are good for binary classification problems, which makes them suitable for fault detection. The downside to SVMs is that when it comes to the case of multi-class classification problems where a data has to be classified either under class A, B or C, the process of classification becomes lengthy and methods like one-against-one or one-against-rest have to be employed. For example in the one-against-rest approach three SVMs have to be trained:

- SVM for class A vs data falling in class B or C (rest)
- SVM for class B vs data falling in class A or C (rest) and
- SVM for class C vs data falling in class B or A (rest)

The number of SVMs to train increase with the number of classes there are. This is not required in the case of neural networks which by default are capable of performing a binary or a multi-class classification task using a single neural network.

3. **Neural Networks for fault detection:**
Constructing a neural network model using the available system measurement data is already explained in the previous section 'modelling of mechanical systems'. Having trained the neural network, any new time-series data point can be fed into the neural network, the neural network provides an estimated output for this input. If the estimated output from the neural network varies from the system's actual output by a value greater then a predefined threshold, the data-point can be classified to belong to the system's faulty operation. The threshold can be fixed or be made to be adaptive and is usually decided based at the modelling error obtained at the time of training the neural network on the measurement data. The value of the threshold impacts the sensitivity with which the faults are detected. Choosing a value for the threshold is important in order to reduce the number of false negatives (high threshold value) and false positives (low threshold value).

2.2.2. Fault Isolation

Fault isolation is defined as the determination of the kind, location and time of detection of a fault. Fault isolation follows fault detection and just like fault detection, it too can be considered to be a classification task where the type of fault which has occurred from a list

of possible faults that can occur for the physical system has to be determined. Such a task requires an insight of the system in order to make a judgment just by looking at the symptoms/faulty measurement data of certain input and output states of the system.
Therefore, the zeroth step, i.e., the step that comes before performing a fault isolation task is to have in place an insight to the system faults. This insight can exist in the form of an inference network that maps a list of possible faults that could occur in the system to the list of symptoms that explain/indicate the occurrence of a particular fault. An example of an inference network for an oil refinery can be seen in figure 2.9, here the top row is a list of the faults while the bottom row contains the symptoms. Hence after a fault has been detected, certain symptoms observed are used to isolate the fault. The inference method is more of a logic based and less of a data based approach especially if the symptoms explaining the faults can be observed visually.

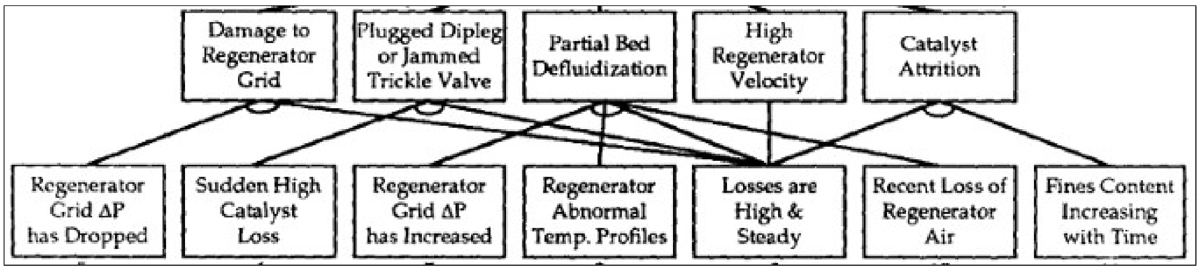

Damage to Regenerator Grid
Plugged Dipleg or Jammed Trickle Valve
Partial Bed Defluidization
High Regenerator Velocity
Catalyst Attrition
Regenerator Grid ΔP has Dropped
Sudden High Catalyst Loss
Regenerator Grid ΔP has Increased
Regenerator Abnormal Temp. Profiles
Losses are High & Steady
Recent Loss of Regenerator Air
Fines Content Increasing with Time

Figure 2.9: The inference network showing a mapping between the faults and symptoms for these faults for an oil refinery [8].

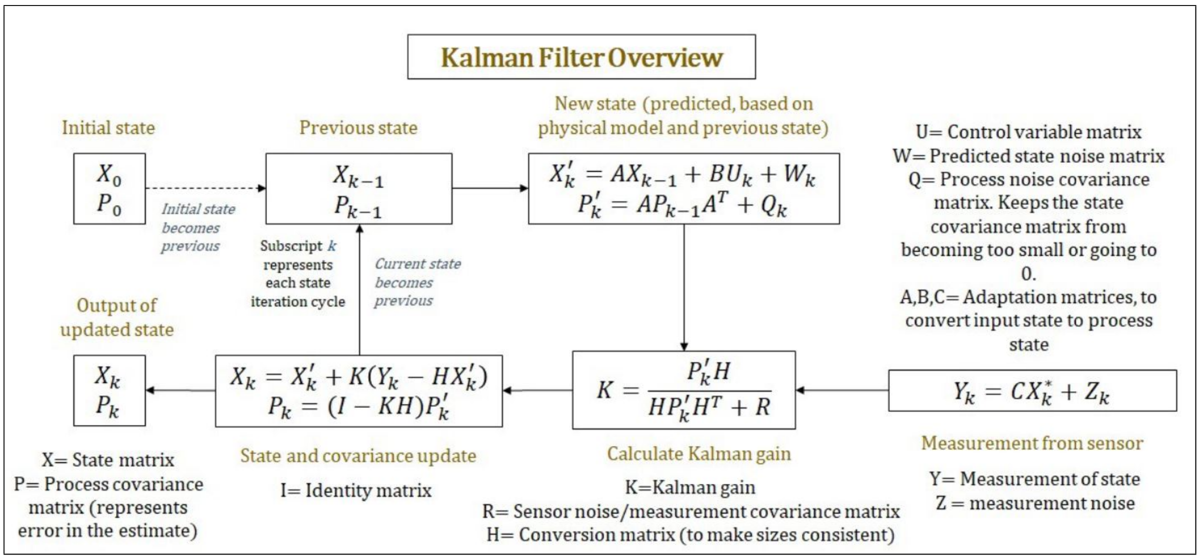
Alternatively, a more data-dependent and popular approach can be used for fault isolation where a classifier of choice can be trained on faulty data that corresponds to a particular type of fault. This training can then be extended to all the fault cases. As a result a trained classifier is obtained which is capable of fault isolation based on the data faulty data that it is provided.
For the data driven approach, as is always the case with classifiers, the training data used to train the classifier must contain combination of unique features that enables the classifier to learn a clear distinction between the different fault cases. To ensure this distinction, before the classifiers are trained, the faulty data corresponding to the different modes of operation of the system are subjected to some signal processing techniques in order to obtain some distinctive features that possibly exist in the data. Some popular techniques applied to the data include Modal Analysis, Fast Fourier Transform (FFT), Wavelet Transform and derivatives and integrals of the system's input states [7].
Although rare, there is a possibility that at any given time instance, multiple faults occurs and it is in situations like these where, the downside of choosing a classifier such as an SVM becomes more apparent because one has to not only take into account to train an SVM for each individual fault case against-the-rest (fault case 1 against-the-rest and so on) but also consider a combination of fault cases against-the-rest (fault case 1 & 2 against-the-rest) and so on.
A final point worth mentioning is that if a binary classifier is already being used for the purpose of fault detection, rather than training yet another classifier for the sole purpose of fault isolation, this binary classifier can be replaced/extended to a multi-class classifier to classify the current operational mode of the system among the various modes (not-fault, fault type 1, fault type 2 and so on) based on the system input data it is provided. Hence in this manner the fault detection and isolation steps can be done in tandem.

2.2.3. Fault Identification

Once a fault in the system has been detected, it is desirable to determine the necessary adjustments to bring the process back to normal operating condition for which a fault identification process is to be carried out. IFAC SAFEPROCESS defines fault identification as the determination of the size and time-variant behaviour of a fault. Fault identification can

be considered to be a reconstruction task where the non-faulty part of the system is to be reconstructed from the faulty system so as to estimate its (non-faulty system) contribution and subtract it from the existing faulty system in order to identify the fault.
Some of the poplar fault identification techniques include:

1. **State-observers**
   State-observers are a popular control-theory and model based approach for fault-identification. The state-observer can be chosen to be a deterministic or stochastic (Kalman Filter and Extended Kalman Filter) depending on the nature of system's dynamics. The operation of state observers for fault identification can be described as follows: At each time step, the estimate of the state $X_k^{'}$ (priori state estimate) is first updated using information about the: 1) State model X_{k-1} (from previous time-step); 2) Control input U_k (at current time-step), and 3) Process covariance matrix P_{k-1} (assumed Gaussian). This estimate is then passed on to a mathematical model (modelling system under normal operation), the model provides the expected system output $X_k^{'}$ which is then compared with the actual output Y of the faulty system. The difference in the two is used to update the priori state estimate $X_k^{'}$ in order to receive the posteriori state estimate X_k. This posteriori estimate is obtained via a gain matrix (Kalman gain matrix) K which is a weightage factor combining the priori state estimate $X_k^{'}$ with the observation differences $Y_k - X_k^{'}$. The gain matrix K itself depends on the state error covariance matrix $P_k^{'}$. Hence, the state observer learns a gain matrix which forces the outputs as predicted by the mathematical model to converge closer to the actual measurements obtained from the faulty system and in this manner the fault identification is carried out. The mathematics behind a state-observer/kalman filter can be visualized in terms of a flowchart in figure 2.10.



Kalman Filter Overview
Initial state
X_0 P_0
Initial state becomes previous
Previous state
X_{k-1} P_{k-1}
New state (predicted, based on physical model and previous state)
$X_k' = AX_{k-1} + BU_k + W_k$
$P_k' = AP_{k-1}A^T + Q_k$
U= Control variable matrix
W= Predicted state noise matrix
Q= Process noise covariance matrix. Keeps the state covariance matrix from becoming too small or going to 0.
A,B,C= Adaptation matrices, to convert input state to process state
Subscript k represents each state iteration cycle
Current state becomes previous
Output of updated state
X_k P_k
$X_k = X_k' + K(Y_k - HX_k')$
$P_k = (I - KH)P_k'$
$K = \frac{P_k' H}{HP_k'H^T + R}$
$Y_k = CX_k^* + Z_k$
X= State matrix
P= Process covariance matrix (represents error in the estimate)
State and covariance update
I= Identity matrix
Calculate Kalman gain
K=Kalman gain
R= Sensor noise/measurement covariance matrix
H= Conversion matrix (to make sizes consistent)
Measurement from sensor
Y= Measurement of state
Z = measurement noise

Figure 2.10: Overview of the working principle and mathematics of a state-observer/Kalman Filter [9].

It should be noted that the state-observer can be supplemented with a Markov Decision Process (MDP) or a Partially Observable Markov Decision Process (POMDP) depending on whether the transitions between states (different operating modes) are either deterministic or stochastic. The dynamics in these discrete states are continuous. Washington in his work [27] shows the implementation of kalman filters with a MDP to carry out the fault detection of a rover. In [26] Jiang et al. carry out a fault identification using state-observes taking into account the time delays present in systems.
Finally it is worth mentioning that the working mechanism of observer-based methods make them suitable candidates for the task of fault detection and fault isolation as well. This can be understood as follows: during the system's normal (no-fault) operating condition, the difference between the state observer's priori state estimate and the system's

actual output $Y - X_k^{'}$ would be negligible as a result of which the difference between the posteriori and priori state estimates $X_k - X_k^{'}$ would also be negligible. However in the occurrence of a fault in the system, the difference in the priori state estimate and the actual systems's output $Y - X_k^{'}$ would be of a considerable magnitude as a result of which the difference between the posteriori and priori state estimates $X_k - X_k^{'}$ would be considerable as well and this indicates a possible occurrence of a fault (fault detection), the system's state for which this difference is large is the location of the fault (fault isolation). This can also be interpreted from the perspective of the gain matrix, any significant change in the gain matrix should indicate a fault in the system.

2. **Least Squares Fitting**
   Least Squares Fitting (LSF) is a model based fault identification method that can be explained with the help of the following example: consider an occurrence of a fault(s) in a physical system, considering that the system is a physical one, the fault can be modelled as an equivalent load(s) (virtual forces and moments) that acts on the undamaged system model to generate a dynamic behaviour identical to the damaged system. The first step is to generate the measurement residuals (example: displacement, velocity, acceleration, etc) by comparing the actual output (faulty system) to the estimated output obtained from the mathematical model of the system (in non-fault condition). The generated residuals are then plugged into the mathematical model of the system (in non-faulty conditions) in order to identify the measured equivalent loads. Meanwhile the theoretical equivalent loads are identified from fault models which are models representing the dynamics of a particular fault in terms of a relationship between fault parameters and the equivalent loads. A least squares algorithm is then implemented in order to best fit between the measured equivalent loads and the theoretical ones obtained from the fault models. In this manner the type, amount and location of the current fault can be estimated. The work by Richard et al. [46] shows the application of the LSF approach to carry out a fault identification task of a faults occurring in rotor systems.

3. **Principal Component Analysis**
   Unlike the previous two methods (LSF and state-observers) which require knowing before hand a white-box model that represents the dynamics of the system, PCA does not require a prior known white-box model. In the described methodology PCA is used for fault identification of faulty sensors as follows: the first step is to carry out a reconstruction of each sensor to determine the non-fault contribution from the faulty error signal. The reconstruction technique is based on PCA. Some popular reconstruction techniques are discussed in detail by Dunia et al. in the following works [28], [47] and [48]. The differences in the reconstruction techniques shown in these papers arise due to a difference in the statistical metrics (SPE, SVI, combined index and T^2) used for carrying out the fault detection task using PCA in each of these work. Following this reconstruction, a set of residuals are defined, these residuals are then monitored to determine a deviation between individual measurements and the normal trend of the remaining variables. The magnitude of the deviation is used to identify the fault.
   The PCA-based scheme mentioned above is based on the assumption that the system follows a linear process. However for the case of nonlinear processes, the linear technique might not perform well, hence there is a need for a framework to help in fault identification for nonlinear systems, this issue is addressed by the use of Kernel PCA methods which can be found in the works of by Choi et al [49], Cho et al. [50] and Carlos et al [51].
   Finally its worth mentioning that the process described above is based on the assumption that at any given time, only one sensor fault can occur. For diagnosing multiple sensor faults which occur at different times, the approach would be to carry out the reconstruction processes in a serial manner.

2.3. Sparse Signal Recovery Problem

A typical Sparse Signal Recovery problem (SSR) can be visualized in figure 2.11. The SSR problem can be considered as a task of solving a linear set of equations formulated in equation 2.3.

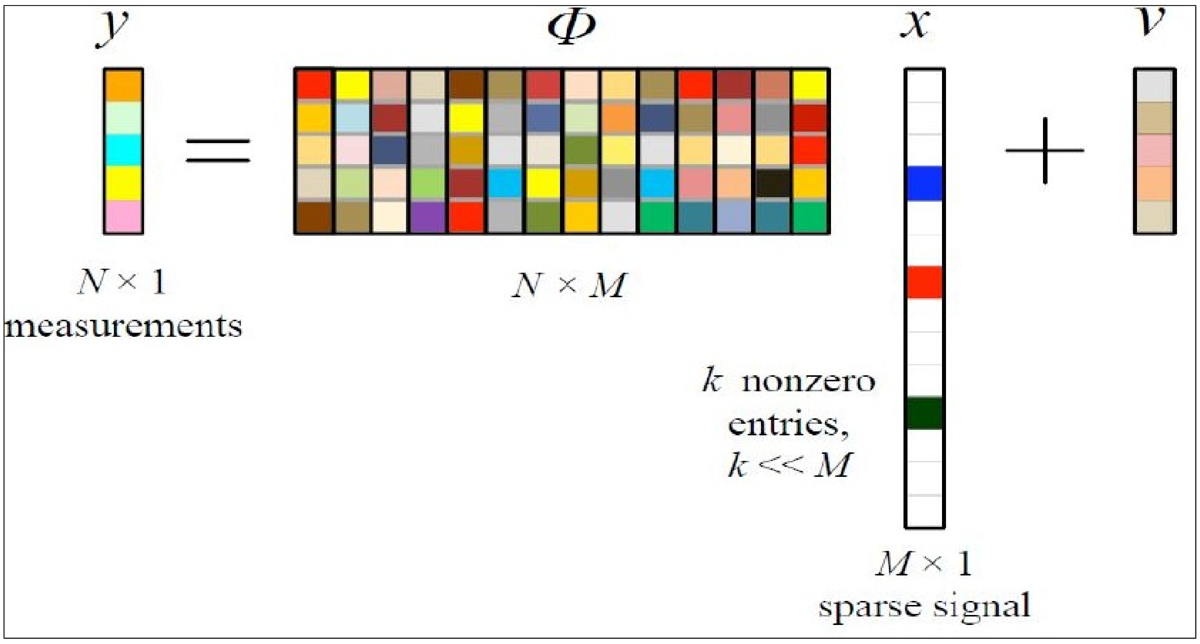


Figure 2.11: Typical structure of a sparse signal recovery problem [10].

$$y = \Phi * x + \gamma \tag{2.3}$$

Where,

- **y** is a '$N \times 1$' vector that represents the system's output for its (system's) single output state for the 'N' different experiments/time-instances. For a system with multiple output states 'J', y would then be a matrix of dimension '$N \times J$' (output for the N experiments across J output states).
- Φ is a matrix of dimension '$N \times M$' that represents the system's input across its (system) 'M' input states for the 'N' different experiments/time-series. 'Φ' is commonly referred to as the dictionary or measurement matrix.
- γ is a '$N \times 1$' vector that represents the measurement noise in the system's single output state for the 'N' different experiments/time-series. The noise is usually modelled as a Gaussian distribution of mean μ and variance σ^2. For a system with multiple output states 'J', 'γ' would then be a matrix of dimension '$N \times J$' (measurement noise for the N experiments across J output states).
- **x** is the '$M \times 1$' k-sparse vector (k non-zero entries in it) that is to be identified. It represents the weights/parameters of the system. For a system with multiple output states 'J', x would then be a sparse matrix of dimension '$M \times J$', where each 'J^{th}' column of x would be a sparse column vector that represents the weights/parameters for the system that corresponds to the 'J^{th}' output state.

The objective in the SSR problem is to solve for the sparse vector 'x' in order to identify the parameters of the model, given the target/output measurement signal 'y' and the dictionary matrix Φ. Mathematically the SSR can be represented in the form of an optimization problem shown in equation 2.4 for the no-noise case and equation 2.5 for the noisy case. The function I in these equations represent the indicator function, whereas β in 2.5 represents a predefined threshold value for the data fit by the model.

$$min_x \sum_{i=1}^{M} I(x_i \neq 0) \textbf{ such that } y = \Phi * x \tag{2.4}$$

$$min_x \sum_{i=1}^{M} I(x_i \neq 0) \textbf{ such that } \|y - \Phi * x\|_2^2 \leq \beta \tag{2.5}$$

Generally the SSR problem is under-determined because the number of model parameters 'M' to be identified is greater than the number of measurements 'N' from the system that are available. For such an under-determined system, there exists infinite such solutions for 'x' that satisfies equation 2.3, however the objective is to identify the sparsest solution 'x' from this set of infinite solutions. Because this system is under-determined, the existence and uniqueness of the solution are guaranteed as soon as the signal is sufficiently sparse, and the measurement matrix satisfies the Restricted Isometry Property (RIP) at a certain level [52].

Before discussing the different sparse recovery algorithms that are used to solve the SSR problem, some applications where solving a SSR problem is required includes:

1. **Speech coding and Compressive sensing:** Large signals are often compressed to a signal with a reduced dimensionality prior being transmitted. After being transmitted, the compressed signal has to be reconstructed to its (signals) original dimension as the signal approaches the receiver's end. This reconstruction requires solving an SSR problem in order to obtain the original decompressed signal.

2. **EEG/MEG:** Different areas of the brain are known to react to different stimulus, keeping this in mind the idea is to provide a known external stimulus and infer the brain activity of different parts of the brain in response to the stimulus by measuring the intensity of the magnetic fields at several locations (points in blue) across the scalp as can be seen in figure 2.12. The objective is to then find the sparse set of active candidate sources (points in red) from the list of all the candidate sources (orange). These active sources indicate the part of the brain that is sensitive to the external stimulus. This process helps in the construction of a map which is of neuro-physiological importance.

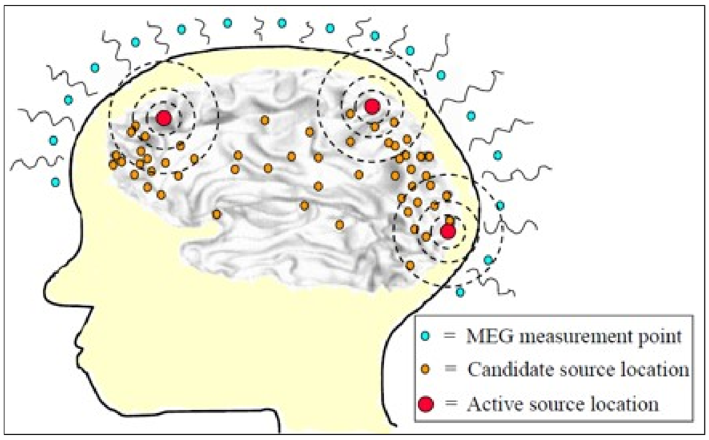

● = MEG measurement point
● = Candidate source location
● = Active source location

Figure 2.12: EEG/MEG setup for inferring brain activity of different parts of the brain in response to stimulus [10].

3. **Modelling of sytems:** The SSR problem can be thought of as a data-driven modelling approach that uses sparse recovery algorithms based on linear regression to identify the model parameters/weights that models the system's input states (values stored in Φ) to the system's output states (values stored in y) from the available time-series measurement data of the system. The sparse recovery algorithms are explained in detailed in the next section of this chapter.

2.3.1. Sparse regression techniques

The SSR problem can be solved by a host of sparse recovery algorithms. These algorithms can be broadly classified (figure 2.13) under three main categories: Greedy methods, Minimization Diversity Measures and Bayesian Methods.

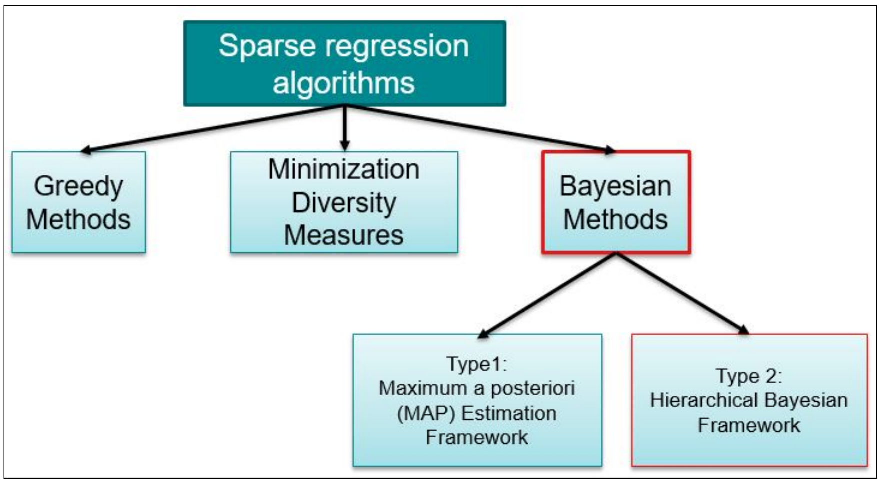

Sparse regression algorithms
Greedy Methods
Minimization Diversity Measures
Bayesian Methods
Type1: Maximum a posteriori (MAP) Estimation Framework
Type 2: Hierarchical Bayesian Framework

Figure 2.13: Overview of the classification of sparse recovery algorithms.

A discussion on these sparse recovery algorithms is provided as follows:

1. **Greedy Methods**
   Greedy methods (figure 2.14) get their name based on the sequential (greedy) manner in which they search for the location of the non-zero elements of the sparse vector. The Matching Pursuit (MP) is a popular greedy solution algorithm proposed by Mallet et al. [53] whose working is described as follows: The first step is to initialize the residual 'r' equal to the measurement vector 't' and then find the first non-zero element 'a' in sparse vector 'w' that best matches the residual 'r' by solving equation 2.6. Once 'a' is identified, for the case when multiple non-zero elements exist in 'w', the next non-zero element 'b' in w can be identified by removing the contribution made by the all previous non-zero elements (here 'a') in w. Removing the contribution of 'a' in w can be done by updating the residual 'r' as per equation 2.7. Having updated the residual 'r', the next non-zero element 'b' is the calculated as per equation 2.6 (replacing 'a' with 'b'). This process is carried out till the residual becomes small enough or after exceeding a pre-defined sparsity threshold.
   A variation to the Matching Pursuit algorithm is the Orthagonal Matching Pursuit (OMP) algorithm [54]. A final note on greedy based methods is that the manner in which

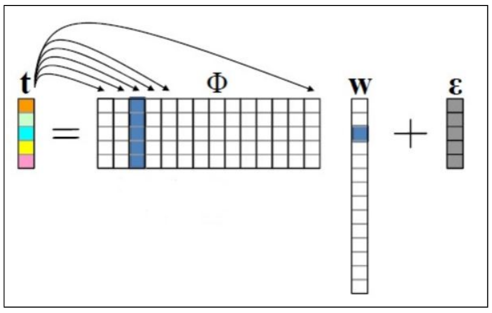


Figure 2.14: Greedy sparse recovery algorithm [10].

$$a = ||\Phi * w - r||_2^2 \tag{2.6}$$

$$r = t - \Phi * w \tag{2.7}$$

2. **Minimization Diversity Measures**
Minimization Diversity Measures are sparse recovery algorithms that solve the SSR problem by defining a cost function and performing an optimization on the defined cost function using methods such as gradient-descent with the objective of minimizing the cost function in order to yield a sparse vector solution to the SSR problem. Choosing a cost function that is continuous and differentiable (smooth) is desirable in order to calculate the gradient of the cost function and carry out the optimization process with more ease.

$$min_w(\|t - \Phi * w\|_2^2 + \lambda \sum_{i=1}^{M} \|w_i\|_p) \tag{2.8}$$

LASSO Regression:

$$min_w(\|t - \Phi * w\|_2^2 + \lambda \sum_{i=1}^{M} \|w_i\|_1) \tag{2.9}$$

Ridge Regression:

$$min_w(\|t - \Phi * w\|_2^2 + \lambda \sum_{i=1}^{M} \|w_i\|_2) \tag{2.10}$$

Equation 2.8 is an example of a cost function that is to minimized in order to yield the desired sparse vector solution 'w'. Here 't' represents the target/measurement and 'Φ' is the dictionary matrix. In this cost function there exists an additional penalty term (right hand side of the addition sign) that allows for a trade-off between model complexity and data fit. Here 'λ' is the regularization term attached to penalty term that regulates this trade off. Assigning λ a large value results in obtaining a cost function where the contribution by the penalty term is significant as a result of which the model complexity increases and the model ends up under-fitting, choosing a lower value of λ results in a minor contribution from the penalty term which results to a lower model complexity and a over-fitting model.
The term 'p' appearing in the cost function in equation 2.8 represents the regularizing norm that is applied to each model parameter 'w_i' of the sparse vector 'w'. Choosing p=1 implies a L1 norm regularization that gives equation 2.9 which is the popular LASSO regression technique [55] that is known to yield a sparse vector 'w'. Whereas choosing p=2 implies a L2 norm regularization that gives equation 2.10 which is another popular regression technique that goes by name of Ridge regression [56]. Unlike LASSO regression, the Ridge regression technique does not yield a sparse vector 'w'.

The reason for obtaining a sparse solution with the L1 norm regularization and a non-sparse solution in the case of the L2 norm regularization can be understood from their mathematical expressions: the L1 norm of any vector X is given by summing up all the absolute values for each element in X. The L2 norm for the same vector 'X' is given by summing up the squares of each element in X and then taking the square root of this summation.
For simplicity, consider a case where the vector X contain two elements 'a' and 'b', let the L1 and L2 norm of vector X calculated as per equations 2.11 (L1 norm) and 2.12 (L2 norm) be 'c_1' and 'c_2' respectively. If all points that have a L1 and L2 norm equal to c_1 and c_2 are plotted graphically then the shape obtained in the case of the L1 norm is a diamond as shown in figure 2.15 (left) where the intercepts at the x and y axes are of magnitude c_1. In the case for the L2 norm the shape obtained is that of a circle as shown in figure 2.15 (right) where the intercepts on the x and y axes are of magnitude c_2. The diamond shape obtained in the case of the L1 norm is likely to intersect the solution hyperplane 'G' at the diamond's tips which yields a sparse solution '$\hat{x}$'. The

circle in the case of the L2 norm is not as likely to intersect the solution hyperplane G at its extreme points but rather at some other point on its circumference which yields a non-sparse solution '$\hat{x}$'.
For a higher dimensional space, the L1 and L2 norms are an octahedron and sphere respectively.

$$\|X\|_1 = |a| + |b| = c_1 \tag{2.11}$$

$$\|X\|_2 = \sqrt{a^2 + b^2} = c_2 \tag{2.12}$$

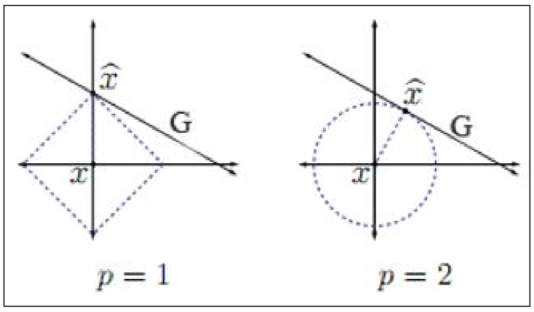


Figure 2.15: Graphical representation of the L1 norm (left) and the L2 norm (right).

3. **Bayesian Methods**
Bayesian Methods make appropriate statistical assumptions on the solution and then apply estimation techniques on the assumptions made in order to yield sparse solutions to the SSR problem. Bayesian Methods can be categorized into Type 1 and Type 2 methods as follows:

   (a) **Type 1: Maximum a Posterior (MAP) Estimation Framework**
   A typical MAP estimation framework can be visualized in figure 2.16. According to the MAP hypothesis, the desired sparse vector $\hat{x}$ is given by the value of x for which the posterior 'P(x|y)' is maximized (equation 2.13). Hence the objective under this MAP estimation framework can be reformulated as: "Finding the 'x' that maximizes the posterior distribution P(x|y)". According to the Bayes theorem, the posterior P(x|y) can be represented as a product of the likelihood 'P(y|x)' with the prior 'P(x)' as shown in equation 2.14. The prior 'P(y)' term is a normalizing term that is constant and hence it can removed from the formulation. The likelihood term 'P(y|x)' is characterized by the noise in the system which is assumed to be a Gaussian and so it remains fixed, this allows the possibility to express equation 2.14 in a closed form as shown in equation 2.15. In equation 2.15, the only selection to be made is that for the prior P(x), which is chosen to be sparse so that the vector x identified by solving the MAP formulation (equation 2.15) is a sparse one. Hence, in this manner the MAP estimation framework has been used to identify the sparse vector x from the known measurements y.
   Choosing different priors P(x) results in obtaining different formulations, for example if P(x) is chosen to be Gaussian then a L2 norm regularization problem (Ridge regression in equation 2.10) is obtained, while choosing P(x) to be a Laplacian results in a L1 norm regularization (LASSO regression in equation 2.9) formulation.
   Alternatively, the concave penalty function g in equation 2.15 can be bounded after which the steps of a Majorize-Minimization (MM) optimizing algorithm [57] can be followed in order to obtain the re-weighted iterative L1 [58] and re-weighted iterative L2 formulations shown in equation 2.16 and equation 2.17 respectively.
   In these re-weighted iterative algorithms: The weights w_i^k are first initialized, this is followed by solving the MAP formulation (equation 2.16 for L1 or equation 2.17 for L2) for obtaining the vector $x_i^{(k+1)}$. The identified vector $x_i^{(k+1)}$ is then used to update the weights from their initialized value w_i^k to their new values $w_i^{(k+1)}$. These new weights $w_i^{(k+1)}$ are then used to solve the MAP formulation once again to obtain the

next estimate of the vector x. This iterative process of solving the MAP formulation and updating the weights continues till the weights have converged. The converged weights are then used to solve the MAP formulation for the final time in order to obtain the required sparse vector x that correspond to the converged weights.

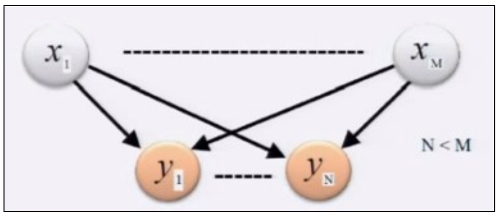


Figure 2.16: MAP estimation framework to identify sparse vector x from known measurements y [10].

According to the MAP hypothesis:

$$\hat{x} = argmax_x P(x|y) \tag{2.13}$$

Expansion according to Bayes rule:

$$\begin{aligned} \hat{x} = argmax_x P(x|y) = argmax_x \frac{P(y|x)P(x)}{P(y)} \\ \hat{x} = argmax_x P(y|x)P(x) \end{aligned} \tag{2.14}$$

Expression in closed form:

$$\begin{aligned} \hat{x} = argmin_x(-\log P(y|x) - \log P(x)) \\ \hat{x} = argmin_x \|y - \Phi * x\|_2^2 + \lambda \sum_{i=1}^{m} g(|x_i|) \end{aligned} \tag{2.15}$$

Re-weighted L1:

$$\begin{aligned} x^{(k+1)} = argmin_x \|y - \Phi * x\|_2^2 + \lambda \sum_i w_i^{(k)} |x_i| \\ x^{(k+1)} = \frac{\partial g(x_i)}{\partial |x_i|} \textbf{ where } x_i = x_i^{(k+1)} \end{aligned} \tag{2.16}$$

Re-weighted L2:

$$\begin{aligned} x^{(k+1)} = argmin_x \|y - \Phi * x\|_2^2 + \lambda \sum_i w_i^{(k)} x_i^2 \\ x^{(k+1)} = \frac{\partial g(x_i)}{\partial x_i^2} \textbf{ where } x_i = x_i^{(k+1)} \end{aligned} \tag{2.17}$$

It is worth noting that although setting up a MAP estimation framework is quite straightforward and essentially only requires a careful selection of the sparse prior P(x) in order to obtain a formulation that can be optimized to obtain the required sparse vector x, practical experience says that performing this optimization on the formulation often results in searching for a solution in a terrain that has several local minima in which the solution is prone to be stuck in. As a result the solution obtained may be close in value to the global minima but may in reality correspond to one of the many local minima.
This downside of the MAP estimation framework motivates the use of Type 2 Bayesian Methods known as the Hierarchical Bayesian Framework which is more commonly known as the Sparse Bayesian Learning (SBL).

(b) **Type 2: Hierarchical Bayesian Framework/Sparse Bayesian Learning**
Tipping in 2001 proposed in his work [59] a Bayesian framework known as sparse Bayesian learning for obtaining sparse solutions to regression problems. A visualization of a typical SBL framework can be seen in figure 2.17. The framework is quite similar in structure to the MAP estimation framework with the exception that an additional vector of hyper-parameters 'γ' have been introduced into the framework in the layer above the sparse vector x. Each hyper-parameter γ_i in this γ layer is connected with the corresponding element in the sparse vector x.
The objective in the sparse Bayesian learning framework is to first identify the optimal values of the hyper-parameters in the γ layer, following which these identified hyper-parameters are used to help identify the required sparse vector x from the measurement y.
In order to estimate the optimal value of the hyper-parameters $\hat{\gamma}$, the MAP hypothesis is applied, which in present context states that: "The desired hyper-parameters $\hat{\gamma}$ is given by the value of γ for which the posterior P(γ|y) is maximized". Mathematically this is given by equation 2.18, which can then be expanded as per Bayes rule to obtain equation 2.20. In equation 2.20 two terms: the likelihood P(x|γ) and the prior P(γ) appear which were obtained by expressing the prior P(x) in the form of a Gaussian Scale Mixture as given in equation 2.19. The likelihood terms P(x|γ) and P(y|x) are defined by the noise in the system which is assumed to be Gaussian. This allows the equation 2.20 to be expressed in closed form as given by equation 2.21. In equation 2.21, the only selection that is to be made is for the prior P($\gamma_i d$), following which the formulation given by equation 2.21 is solved in order to obtain the optimal value of the hyper-parameters $\hat{\gamma}$. Once identified, these hyper-parameters γ are then fixed and the posterior P(x|y;γ) is calculated by solving a MAP formulation between x and y (as was shown in the MAP estimation framework sub-section) to obtain the required sparse vector x from measurements y.
SBL also has its re-weighted iterative L1 and L2 algorithm variants. In the SBL variants, the iterative process is slightly different: At first the weights are initialized, these weights are then used to solve a MAP formulation to obtain sparse vector x. The sparse vector x is then used to find the first set of values for the hyper-parameters γ. These hyper-parameters are then used to update the weights from their initialized value to their new values for the next iteration. These updated weights are then used to solve the MAP formulation to obtain the new sparse vector x for the next iteration. This iterative process continues till the hyper-parameters γ converges or a stopping criterion is reached. The vector x that corresponds to the converged hyper-parameter γ_i is the required sparse vector.

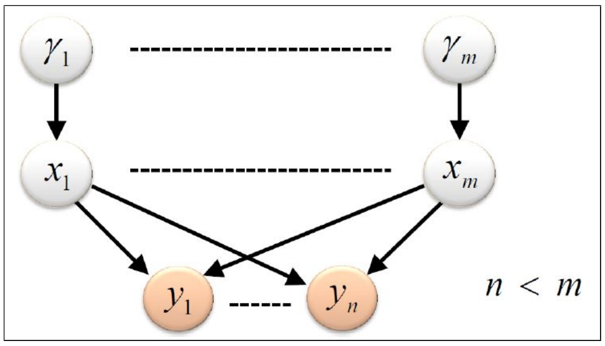


Figure 2.17: Typical Sparse Bayesian Learning framework with a list of hyper-parameters γ, corresponding elements in sparse vector x and available measurements y [10].

According to the MAP hypothesis:

$$\hat{\gamma} = argmax_{\gamma} P(\gamma|y) \tag{2.18}$$

Expansion of the prior P(x) represented in the form of a Gaussian Scale Mix-

ture:

$$\textbf{Separability: } P(x) = \Pi_i P(x_i)$$
$$P(x_i) = \int P(x_i|\gamma_i)P(\gamma_i)d\gamma_i \tag{2.19}$$

Expansion of equation 2.18 **according to Bayes rule and substituting equation** 2.19**:**

$$\hat{\gamma} = argmax_\gamma P(\gamma|y) = argmax_\gamma \int P(y|x)P(x|\gamma)P(\gamma) \tag{2.20}$$

Expression in closed form:

$$\hat{\gamma} = log|\mu_y| + y^T \mu_y^{-1} y - 2\sum_i \log P(\gamma_i) \tag{2.21}$$

Note: *The presentation by Bhaskar Rao [10] on Sparse Bayesian Learning can be referred to for a more elaborate explanation of the mathematics used in both Type 1 and Type 2 methods.*

Although the SBL and the MAP estimation approach have many similarities in their respective frameworks with only exception being that the SBL approach introduces a layer of hyper-parameters γ in its framework, practical experience shows that using the SBL framework results in searching for a solution in a terrain with much fewer local minima than the MAP estimation approach. This means that the chances of the SBL approach getting stuck in the local minima is lower which increases the chance that the sparse solution obtained corresponds to the global minima. This can be thought of intuitively as: "By using a SBL framework, we move up one hierarchy level and hence move further away from the measurement data y as a result of which the terrain gets a bit blurred. This results in several of the local minima to become hidden. What is left on the terrain now are the main peaks which are more likely to be captured in comparison to the local minima."

The motivation to choose a SBL approach over other techniques such as Greedy methods and Minimization diversity measures for solving a SSR problem can be summarized in the following points:

- SBL algorithms have not only shown to outperform other algorithms in practical applications, but have also solved some problems where other methods have either failed or have been unable to provide a reliable solution [60]. The work by Zhang et al. in [61] is one such example where a variant of SBL known as the Block Sparse Bayesian Learning (BSBL) approach was applied for reconstruction of FECG recordings.
- In general the SBL approach shows a better performance in terms of local and global convergence than other methods [60]. The work of Youness et al. [62] can be referred to for a comparison of the performance of six different sparse recovery methods belonging to Greedy, Minimization Diversity and Bayesian Methods.
- SBL's recovery performance is robust to the characteristics of the dictionary matrix (Φ) [60], i.e., even when the columns of this matrix are highly coherent, SBL manages good performances while other algorithms show poor performance. The work by David [63] shows the SBL approach outperforming Type 1 Bayesian methods for a highly structured (correlated) dictionary matrix for the application of MEG source localization.

Although the SBL approach has many advantages over other sparse recovery approaches, one drawback of the SBL approach is that it is generally computationally expensive to solve in comparison to the other methods. The computational loads increase with the increase in the number of parameters that are to be estimated.

2.4. Discussion

In this chapter, concepts and methods from the domains of: modelling of systems, fault diagnosis and sparse signal recovery were extensively discussed. Although neural networks by themselves cannot perform the fault identification task, if supplemented with a sparse recovery that can perform a sparse reconstruction of the neural network's weights from the available time-series measurement data, then in theory a fault identification could be carried out. Finally, it was discussed that among the recovery methods, the sparse Bayesian learning approach seems to outperform other methods in finding an accurate sparse solution.

3
Fault Detection, Isolation and Identification

In the previous chapter, different methods for carrying out fault detection, isolation and identification ware discussed. This chapter proposes and describes the fault diagnosis algorithms, namely fault detection and fault isolation & identification algorithms that are used to carry out a fault diagnosis in this study.

3.1. Fault Detection Algorithm

The fault detection algorithm is shown in figure 3.1. At time-step 't+1', the inputs states are simultaneously provided to the system, the neural network modelling the system and the adaptive threshold model (explained ahead). The neural network estimates the system's output 'y_m' for time-step 't+1'. The difference between the expected output (y_m) and actual output (y_t) of the system is used to generate the residual 'r' at time-step 't+1'. This residual is compared to the upper (T_u) and lower (T_l) threshold values provided by the threshold model for time-step 't+1'. If the residual is greater than or equal to the upper threshold value ($r \geq T_u$), or less than or equal to the lower threshold value ($r \leq T_l$), the fault detection algorithm considers that a fault may have occurred (not yet classified as fault yet) at time-step 't+1'. If this condition is not met, then the algorithm classifies the system at time-step 't+1' as not a fault. This process is carried out at each time-step during the operation of the system.

The reason that the system is not directly classified to be at fault when the fault detection condition ($r \geq T_u$ or $r \leq T_l$) is satisfied for time-step 't+1' is because there is a possibility that time-step 't+1' is a single point outlier i.e. false positive which may have occurred as a result of noise or inaccuracies in modelling the system. In order to prevent over-sensitivity to outlier(s) and detect true occurrence of fault(s), the successive time-steps after 't+1' ('t+2', 't+3', and so on) are checked against the fault detection condition ($r \geq T_u$ or $r \leq T_l$). If for most of these successive time-steps the condition is met, then the fault-detection algorithm classifies the occurrence of a fault at time-step 't+1'. Conversely, if the condition is not true for the majority of successive time-steps, then it is highly likely that time-step 't+1' is a single point outlier and so the algorithm classifies the system at time-step 't+1' as not a fault.

3.1.1. Adaptive Threshold

The threshold is of key importance in detecting the occurrence of a fault in this algorithm. Selecting a suitable threshold is a challenging task. A small threshold value will result in false positives i.e. data points belonging to normal operation being classified as a fault. Whereas a large threshold will result in false negatives i.e, data points that actually belong to the system's faulty operation being classified as not-fault. Although it is desirable for the fault

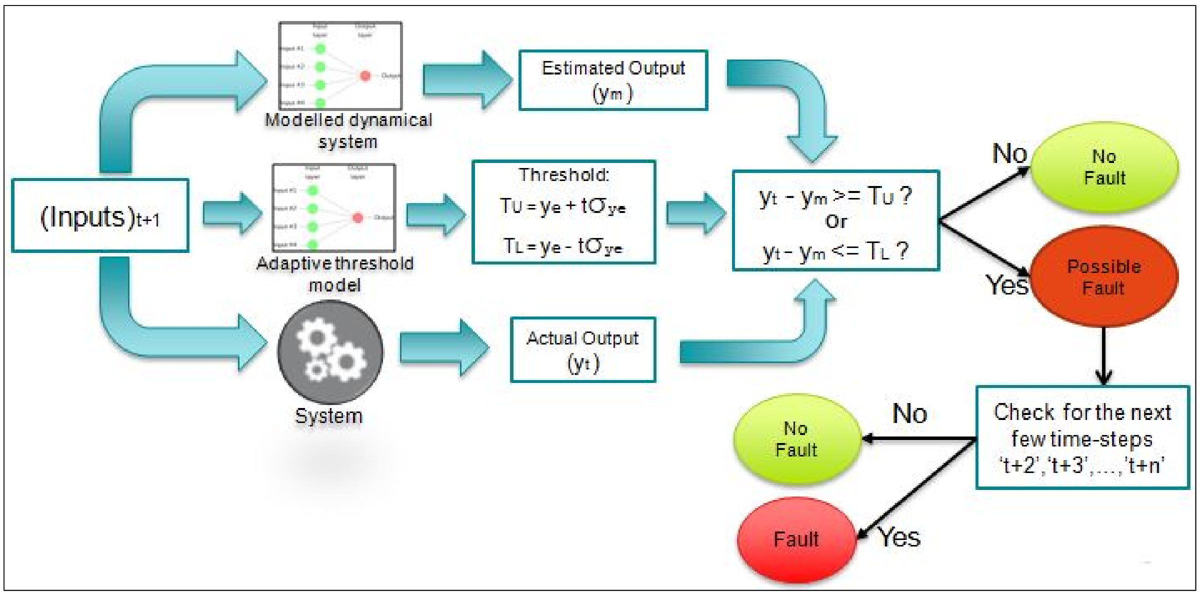


Figure 3.1: Fault detection algorithm.

detection algorithm to correctly classify each time-step data point, in practice achieving a perfect classification is nearly impossible. Therefore the goal is to have a threshold that can help achieve a good classification accuracy thereby resulting in a reduced number of false positives and false negatives while not memorizing (over-fitting) the data in order to achieve such result.

The threshold can either be static or adaptive. A static threshold is one that remains fixed in value through-out the operation of the system. While suitable in some applications, static threshold can be disadvantageous for fault detection when a fault is not characterized solely on the magnitude of the residual but rather its magnitude in relation to the input states of the system at that time-step. For such a system, implementing a static threshold would result in an increased number of incorrect classifications. The alternative is implementing an adaptive threshold whose value changes at each time-step during the system's operation and is provided by a model that has been trained in prior to identify a relation between the input-residual pairs of the system. This naturally results in a reduction of the number of incorrect classifications by the fault detection algorithm. An example for the comparison between a static and an adaptive threshold can be seen in figure 3.2.

Assuming that the system is modelled by a neural network, the design procedure for identifying an adaptive threshold model [64] for assisting the fault detection algorithm is described as follows:

- Residuals 'r_i' are generated ($r_i = y_t - y_m$) for each data point 'i' in the training data set. Here y_t and y_m are the target output and estimated output for the i^{th} data point in the training data set, respectively. The input states of the system 'u_i' and the corresponding residual r_i are then paired and stored in a separate data set $(u_i, r_i)_{i=1}^{N}$.

- A new neural network is trained on the input-residual pairs data set. The newly identified model constitutes an estimate of the error due to under-modelling (neural network modelling the system), and is called the error or adaptive threshold model.

Once trained, the adaptive threshold model is ready for implementation in the fault detection algorithm as follows: at time-step 't' of the system's operation, the upper (B_u) and lower confidence bounds (B_l) are calculated as per equations 3.1 and 3.2. Here T_u and T_l are the upper and lower thresholds calculated by the adaptive threshold model as per equations 3.3

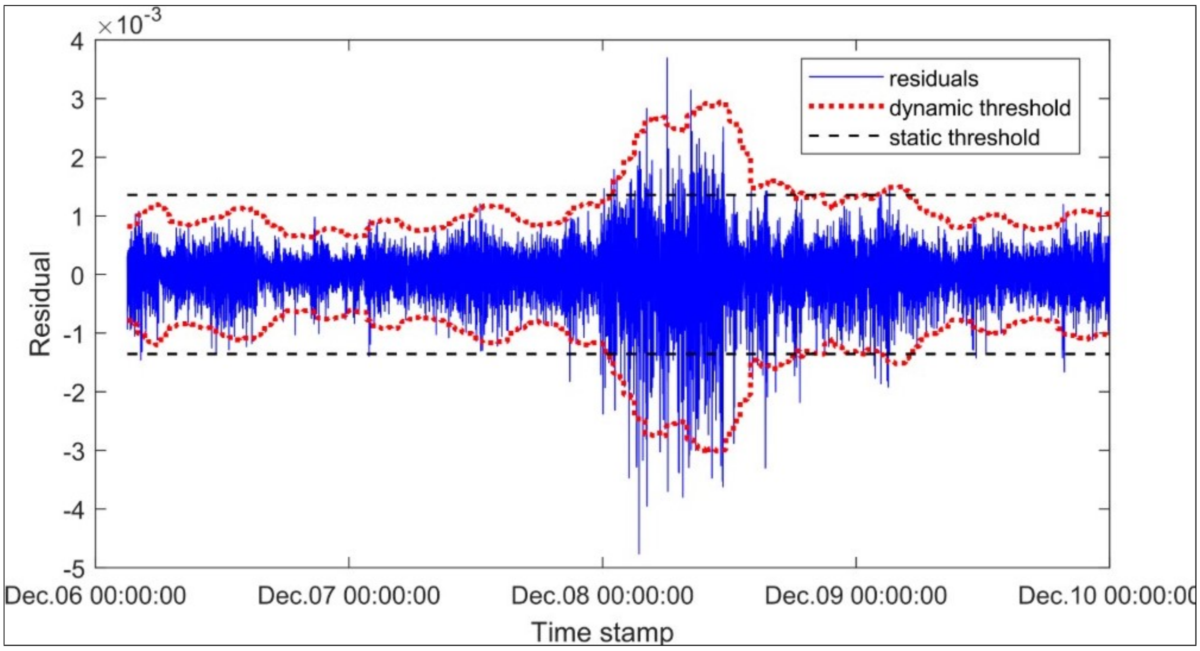


Figure 3.2: Comparison between a static and an adaptive/dynamic threshold for the purpose of a fault detection based on the residual generation.

and 3.4, respectively. The 'v' term in equations 3.3 and 3.4 is the standard deviation of y_e while t_β is the $\mathcal{N}(0,1)$ tabulated value assigned to a given confidence level.

$$B_u = y_m + T_u \tag{3.1}$$

$$B_l = y_m + T_l \tag{3.2}$$

$$T_u = y_e + t_\beta v \tag{3.3}$$

$$T_l = y_e - t_\beta v \tag{3.4}$$

3.2. Fault Isolation and Identification Algorithm

Once the fault detection algorithm has detected an error at time-step 't+1', the next task is to carry out the fault isolation and identification. The fault isolation and identification tasks are performed in tandem by applying the sparse Bayesian learning approach to first reconstruct the weights between successive layers of the neural network (fault identification) and from these reconstructed weights, identify those weights that have changed in their value and isolate them (fault isolation).
This section covers the fault isolation and identification for the following three cases:

- Case 1: Linear systems modelled by neual networks without a hidden layer.
- Case 2: Linear systems modelled by neural networks having a hidden layer.
- Case 3: Nonlinear systems modelled by neural networks having a hidden layer with a nonlinear activation function.

A separate approach for linear and non-linear systems arise due to the difference in the neural network architecture required to model them. A linear system can be modelled by a neural network architecture that has an input and an output layer but may or may not have a hidden layer. Whereas, to model a nonlinear system, the neural network architecture must have at least 1 hidden layer (with nonlinear activation function on this layer).

Note: for the fault detection task, separate approaches were not required because the fault detection algorithm is independent of the neural network architecture and whether or not the system is linear or nonlinear.

The fault isolation and identification algorithms for the three cases are described below:

- **Case 1: Fault isolation and identification for linear systems modelled by neural networks without a hidden layer**
  Consider a generic linear system that has six input states $x_1, x_2, x_3, x_4, x_5, x_6$ and two output states y_1, y_2. Let the normal operating dynamics of the linear system be described by equation 3.5, where θ_1 to θ_{12} are fixed coefficients of the input states and γ_1, γ_2 are measurement noise that is assumed to be Gaussian.

$$\begin{aligned} y_1 &= \theta_1 * x_1 + \theta_2 * x_2 + \theta_3 * x_3 + \theta_4 * x_4 + \theta_5 * x_5 + \theta_6 * x_6 + \gamma_1 \\ y_2 &= \theta_7 * x_1 + \theta_8 * x_2 + \theta_9 * x_3 + \theta_{10} * x_4 + \theta_{11} * x_5 + \theta_{12} * x_6 + \gamma_2 \end{aligned} \tag{3.5}$$

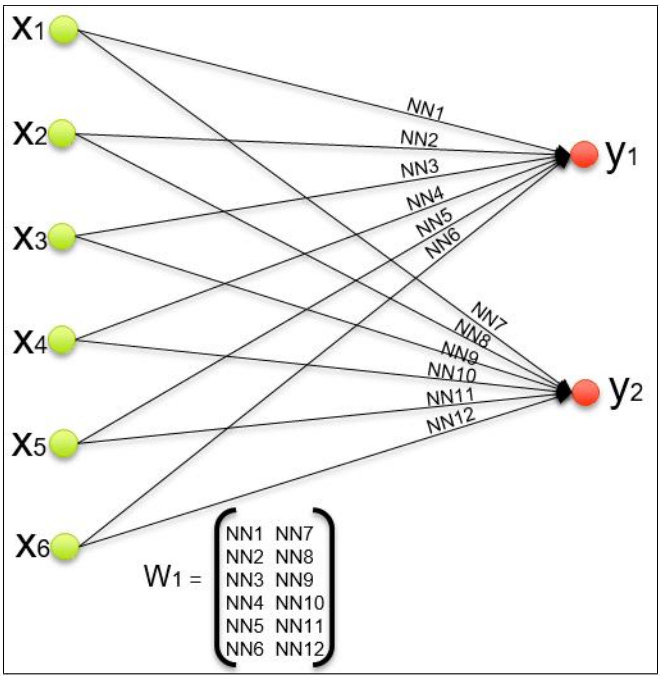


Figure 3.3: Neural network architecture without a hidden layer used for modelling the linear system.

Let the system be modelled by a neural network that does not have a hidden layer, as shown in figure 3.3. In this figure the green and red neurons represent the input and output neurons in the input and output layers, respectively. NN_i represent the weights of the neural network, with NN_1 to NN_6 connecting the six input neurons to the first output neuron (y_1), and NN_7 to NN_{12} connecting them to the second output neuron (y_2). Finally, W_1 represents all the set of weights in the neural network in a matrix form. Once trained, the system dynamics can be represented in terms of the weights of the neural network as per equation 3.6.

$$\begin{aligned} y_1 &= NN_1 * x_1 + NN_2 * x_2 + NN_3 * x_3 + NN_4 * x_4 + NN_5 * x_5 + NN_6 * x_6 \\ y_2 &= NN_7 * x_1 + NN_8 * x_2 + NN_9 * x_3 + NN_{10} * x_4 + NN_{11} * x_5 + NN_{12} * x_6 \end{aligned} \tag{3.6}$$

Assume that at time-step 't+1' during the system's operation a fault occurs, as a result of which the system's properties change. This can be related to a change in the values of certain weights in the neural network modelling the system. Let NN_1 and NN_8 be the changed/faulty weights which after the occurrence of the fault change their values to $NN_1^{'}$ and $NN_8^{'}$, respectively. The dynamics of the faulty system expressed in terms of the changed neural network's weights is given by the equation 3.7.

$$\begin{aligned} y_1 &= NN_1^{'} * x_1 + NN_2 * x_2 + NN_3 * x_3 + NN_4 * x_4 + NN_5 * x_5 + NN_6 * x_6 \\ y_2 &= NN_7 * x_1 + NN_8^{'} * x_2 + NN_9 * x_3 + NN_{10} * x_4 + NN_{11} * x_5 + NN_{12} * x_6 \end{aligned} \tag{3.7}$$

Following the detection of this fault by the fault detection algorithm, in order to reconstruct the weights of the neural network (fault identification), enough number of input and output measurement data from the operating faulty system have to be recorded. Practical experience says that an accurate reconstruction of the weights requires at-least twice the number of observations as the number of weights that are to be determined. If the number of measurements available are less, then the accuracy with which the weights are reconstructed decreases.
The task of reconstructing the weights from the collected measurement data set can be reformulated into the task of solving a sparse signal recovery (SSR) problem by using the sparse Bayesian learning approach as shown in figure 3.4.

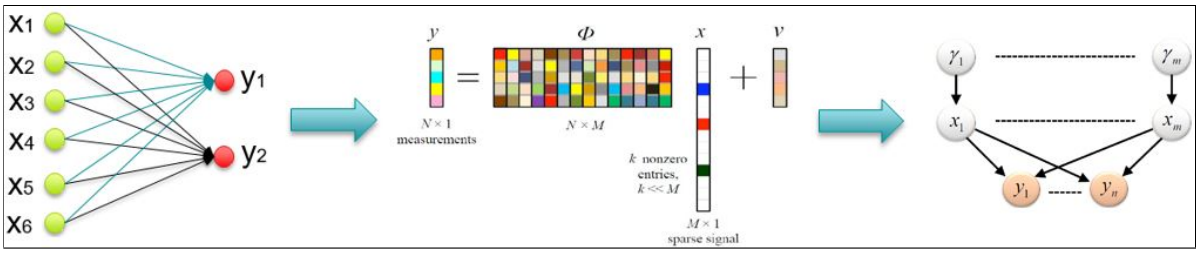


Figure 3.4: Reconstruction of neural network weights by reformulating into a sparse signal recovery problem.

In this formulated SSR problem, y is a vector of dimension M×1 which represents the system's output state y_1 across M measurements recorded from the faulty system. The dictionary matrix Φ is of dimension M×6 where each column represents one of the six input states (x_1 to x_6) of the faulty system across the M measurements recorded from this faulty system. The vector x of dimension 6×1 represents the six reconstructed weights $NN_1^{'}$ to $NN_6^{'}$ that are identified upon solving the SSR problem. The reconstruction of these six weights is followed by solving another SSR problem for the reconstruction of the next six weights $NN_7^{'}$ to $NN_{12}^{'}$. In the second SSR problem, the dictionary matrix Φ remains the same and vector y now represents the output state y_2 across the same M measurements.

The re-weighted L_1 type sparse Bayesian learning algorithm [60] that is used to solve the SSR problem is shown in algorithm 1. Recall from the previous chapter the working of the iterative SBL algorithm: the vector of reconstructed weights x^{k+1} from the k^{th} iteration is used to update the value of the hyper-parameters γ_i^{k+1} in the k^{th} iteration. These hyper-parameters are in turn used to update the weights w_i^{k+1} in the k^{th} iteration, and finally, these weights are used to calculate the vector of reconstructed weights x^{k+2} in the $(k+1)^{th}$ iteration. This iterative process continues till the hyper-parameters γ_i^{k+1} converge or a stopping criterion is reached. The vector x that corresponds to the converged hyper-parameter γ_i is the required set of reconstructed weights $NN_1^{'}$ to $NN_6^{'}$ and $NN_7^{'}$ to $NN_{12}^{'}$.

This algorithm requires the calculation of the hessian matrix H which in the case of the neural network is the double derivative of the cost/error function E with respect to the neural network's reconstructed weights $\left(\frac{\partial^2 E}{\partial (NN^{'})^2}\right)$. In algorithm 1, the hessian matrix H is expected to be of dimension 6×6 and its expression is given in equation 3.8. The work by C.Bishop [65] covers the calculation of the hessian matrix for a multi-layered

Algorithm 1 Re-weighted L_1 type sparse Bayesian learning algorithm for reconstruction of the neural network weights (fault identification) for the linear case.

```
1:  Fault Identification by weight reconstruction:
2:  Initialize $\lambda = 1$ and $w_i^1$ as unit vector;
3:  Assign B to be an identity matrix of dimension 6×6; B = eye(6);
4:  for $k = 1, \ldots, k_{max}$ do
5:      $x^{k+1} = argmin_x(\|y - \Phi x\|_2^2 + \lambda \sum_{i=1}^{6} \|w_i^k .* B_{i,:}x\|_1)$ ;
6:      $\gamma_i^{k+1} = \left|\frac{B_{i,:}x^{k+1}}{w_i^k}\right|$ ;
7:      $\tau^{k+1} = diag(\gamma_i^{k+1})$ ;
8:      $C^{k+1} = (B^T(\tau^{k+1})^{-1}B + H)^{-1}$ ;
9:      $\alpha_i^{k+1} = -\frac{B_{i,:}C^{k+1}B_{i,:}^T}{(\gamma_i^{k+1})^2} + \frac{1}{\gamma_i^{k+1}}$ ;
10:     $w_i^{k+1} = \sqrt{\alpha_i^{k+1}}$ ;
11:     if $\gamma_i^{k+1}$ has converged or a stopping criterion has been reached then
12:         break ;
13:     end if
14: end for
15:
16:
17: Where $B_{i,:}$ stands for the $i^{th}$ row of the matrix B
18: Where $H$ stands for the hessian matrix
19: Where $.*$ represents dot product
```

neural network.

$$H = \begin{bmatrix} \frac{\partial^2 E}{\partial (NN_1')^2} & \frac{\partial^2 E}{\partial NN_1' \partial NN_2'} & \cdots & \frac{\partial^2 E}{\partial NN_1' \partial NN_6'} \\ \frac{\partial^2 E}{\partial NN_2' \partial NN_1'} & \frac{\partial^2 E}{\partial (NN_2')^2} & \cdots & \frac{\partial^2 E}{\partial NN_2' \partial NN_6'} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial NN_6' \partial NN_1'} & \frac{\partial^2 E}{\partial NN_6' \partial NN_2'} & \cdots & \frac{\partial^2 E}{\partial (NN_6')^2} \end{bmatrix}$$
$$H = \begin{bmatrix} \frac{\partial^2 E}{\partial (NN_7')^2} & \frac{\partial^2 E}{\partial NN_7' \partial NN_8'} & \cdots & \frac{\partial^2 E}{\partial NN_7' \partial NN_{12}'} \\ \frac{\partial^2 E}{\partial NN_8' \partial NN_7'} & \frac{\partial^2 E}{\partial (NN_8')^2} & \cdots & \frac{\partial^2 E}{\partial NN_8' \partial NN_{12}'} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial NN_{12}' \partial NN_7'} & \frac{\partial^2 E}{\partial NN_{12}' \partial NN_8'} & \cdots & \frac{\partial^2 E}{\partial (NN_{12}')^2} \end{bmatrix} \tag{3.8}$$

Following this reconstruction, the equations for the faulty system can be written in terms of the reconstructed weights, as shown in equation 3.9.

$$\begin{aligned} y_1 &= NN_1' * x_1 + NN_2' * x_2 + NN_3' * x_3 + NN_4' * x_4 + NN_5' * x_5 + NN_6' * x_6 \\ y_2 &= NN_7' * x_1 + NN_8' * x_2 + NN_9' * x_3 + NN_{10}' * x_4 + NN_{11}' * x_5 + NN_{12}' * x_6 \end{aligned} \tag{3.9}$$

The reconstructed weights (NN_i') that are not equal in value to their corresponding initial weights (NN_i) imply that these weights changed their value after occurrence of the fault and hence are to be isolated. In this example, only the reconstructed weights NN_1' and NN_8' are expected to differ in value from their initial values NN_1 and NN_8, implying that these weights were affected by the fault and are to be isolated. The fault isolation algorithm can be seen in algorithm 2.

Algorithm 2 Fault isolation algorithm for the reconstructed neural networks

```
1: Fault Isolation:
2: for $z = 1, .., 12$ do
3:     if $NN_z' \neq NN_z$ then
4:         Isolate this weight;
5:     end if
6: end for
```

- **Case 2: Fault isolation and identification for linear systems modelled by neural networks having a hidden layer**
  Consider the same generic linear system from case 1 whose system dynamics are given by equation 3.5. Let the system be modelled by a neural network architecture having a hidden layer, as shown in figure 3.5. The purple neurons in the figure represent the neurons in hidden layer. Once trained, the system dynamics can be represented in terms of the weights of the neural network as per equation 3.10 or in the matrix form as given in equation 3.11.

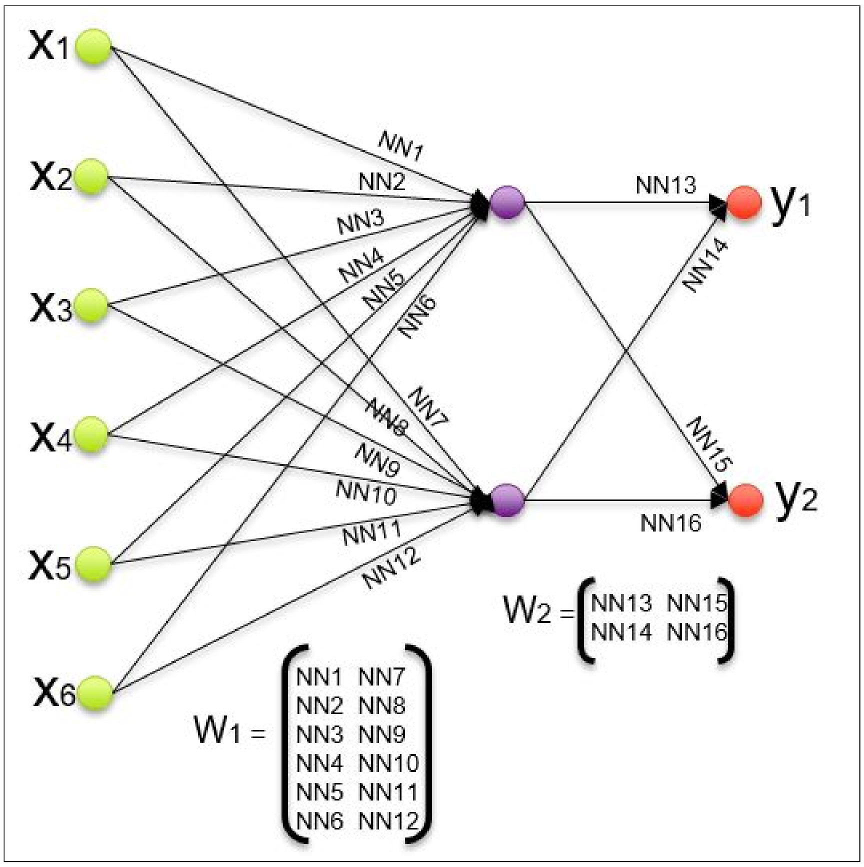


Figure 3.5: Neural network architecture with a hidden layer.

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = \left(\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{bmatrix} \times \begin{bmatrix} NN_1 & NN_7 \\ NN_2 & NN_8 \\ NN_3 & NN_9 \\ NN_4 & NN_{10} \\ NN_5 & NN_{11} \\ NN_6 & NN_{12} \end{bmatrix} \right) \times \begin{bmatrix} NN_{13} & NN_{15} \\ NN_{14} & NN_{16} \end{bmatrix} \tag{3.10}$$

$$Y = f(X \times W_1) \times W_2 \tag{3.11}$$

$$f(Z) = Z \tag{3.12}$$

Here f is a linear activation function (equation 3.12) applied on the neurons in the hidden layer. Finally, W_1 and W_2 represent the set of weights connecting the input layer to the hidden layer and the hidden layer to the output layer of the neural network, respectively.
In the previous case (figure 3.3), implementing the sparse Bayesian learning algorithm on the system's neural network architecture was straightforward because the neural network did not have a hidden layer resulting in a direct mapping between the neurons of the input layer to the neurons of the output layer. However, in case 2, the presence of a hidden layer (figure 3.5) prevents a direct mapping between the neurons of the input layer to the neurons of the output layer. As a result, the weight reconstruction process has to be carried out over two stages. The first reconstruction is for the set of weights lying between the input and hidden layer i.e. W_1 and the second reconstruction is for the set of weights lying between the hidden and output layer i.e. W_2.
However, this stage-wise reconstruction process is not straightforward. Take for example the reconstruction of the set of weights in W_1 which requires solving a SSR problem formulated between the input and hidden layer's neurons. The issue with solving this SSR problem arises with a realization that although the values of neurons in the input layer are known (time-series measurements corresponding to input states of the faulty system), the values of the neurons in the hidden layer are unknown. Therefore, in the SSR problem formulation, the dictionary matrix Φ of dimension M×6 can be formed (from the six input states measurement data) but the vector y of dimension M×1 can not be formed (hidden layer's neuron values are not known). Hence, this results in an incomplete SSR problem formulation. Similarly, the reconstruction of weights in W_2 requires solving a SSR problem formulated between the neurons of the hidden and output layers. Although the values of the neurons in the output layer are known (time-series measurements corresponding to output states of the system), those of neurons in the hidden layer are unknown. Therefore, in the SSR problem formulation, the measurement vector y of dimension M × 1 can be formed (from output state(s) measurement data) but the dictionary matrix Φ of dimension M × 2 can not be formed (hidden layer's neuron values are not known). Hence, in order to carry out stage-wise reconstruction, the first step is to estimate the values of neurons in the hidden layer.

This estimation relies on making an educated guess on the possible locations of the weights in the neural network that have changed in value after the occurrence of the fault. For this, a new set of neural networks are trained from the inputs states of the faulty system onto the prediction errors (actual system output - neural network predicted output) at the output states. The idea behind training these new neural networks is to find out which weight(s) of the neural network modelling the system can be held accountable (indicated by a pattern in the values of the weights of the newly trained neural networks) for the prediction error observed at the output states. The architecture of these new neural networks can be seen in figure 3.6. The input layer consists of 6 neurons which correspond to each input state (x_1 to x_6) of the system and the output layer consists of one neuron that represents the prediction error at the output states (y_1 and y_2) of the system.

Practical experience shows that distinctive patterns in values of the weights of the newly trained neural networks are observed depending on the true location(s) of the changed-weight(s) in the neural network modelling the system. Recognizing these patterns help narrow down the possible locations of the changed weights from many to only a few in number. These distinctive patterns are described below:

- **Situation 1: True location of changed weight(s) lies between the input and hidden layers (W_1) only**

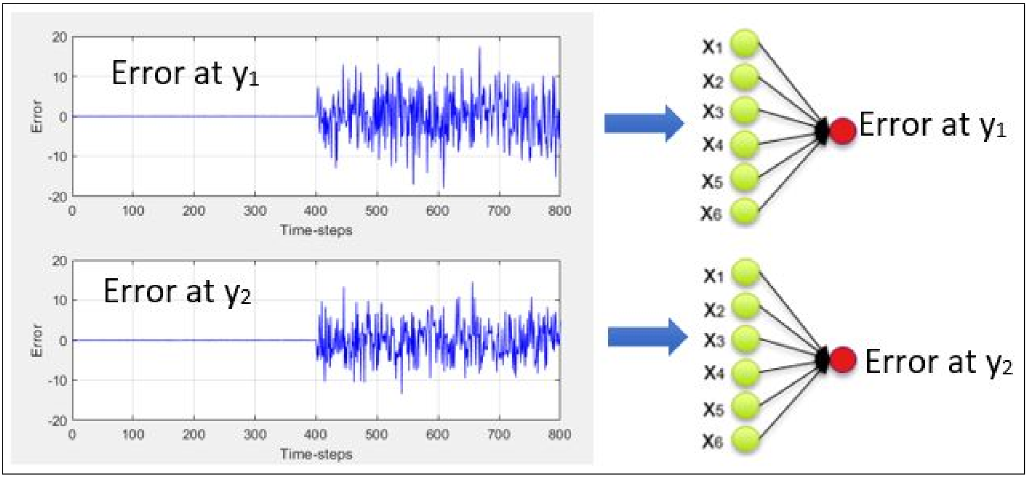


Figure 3.6: New neural networks (right) trained on the prediction errors (left) at the output states.

Consider that a fault in the linear system results in changing of one of the neural network's weights that lies between the input and hidden layers (W_1). Assume this weight to be NN_1, which after the occurrence of the fault changes its value to NN_1'. Training two new neural networks (figure 3.7 (left)) on the prediction errors for output states (y_1 and y_2) reveals a distinctive pattern in the values of the trained weights, as listed in table 3.1 (situation 1 example 1). In both the newly trained neural networks, the first weight (in red) has a large magnitude while the other weights have a low or nearly zero magnitude. From this pattern, an educated guess can be made that the changed weight in the neural network modelling the linear system belongs to the set of weights in W_1. It's location can be further narrowed down to either one of the weights going from the first input neuron to the rest of the neurons in the hidden layer (NN_1 or NN_7), as shown by the red dotted weights in figure 3.7 (right). Similarly, when multiple weights in W_1 change, for example NN_1 and NN_{10}, the pattern observed in the trained weights is listed in table 3.1 (situation 1 example 2). From this pattern, again an educated guess can be made that all weights other than the red (NN_1 and NN_7) and blue (NN_4 and NN_{10}) dotted weights in figure 3.8 (right) remain unchanged.

Neural Network #	Weight #	Situation 1: Changed weight located b/w input and hidden layer only (W1 only)		Situation 2: Changed weight located b/w hidden and output layer only (W2 only)	Situation 3: Changed weight located in both W1 and W2 layers
		Example 1	Example 2	Example 1	Example 1
Neural Network 1 trained on prediction error at output state 1	NN 1	6.4295	-8.6014	-4.5536	9.6598
	NN 2	-0.0063	-0.1051	2.6786	-0.0260
	NN 3	-0.0074	-0.0188	3.5804	0.1364
	NN 4	-0.1757	-10.5946	-5.4241	0.0021
	NN 5	-0.0085	-0.0575	-1.8455	0.0115
	NN 6	-0.0188	-0.0188	-3.6033	0.1352
Neural Network 2 trained on prediction error at output state 2	NN 1	3.7989	-3.8339	N/A	-11.4998
	NN 2	-0.0260	0.0062	N/A	-1.8901
	NN 3	-0.0220	0.0332	N/A	-2.8103
	NN 4	-0.1234	-4.9218	N/A	6.4595
	NN 5	-0.0182	0.0101	N/A	1.4260
	NN 6	-0.0043	0.0132	N/A	4.5868

Table 3.1: Linear systems: Distinctive patterns in the values of the weights of the new neural networks trained on the prediction errors of system's output states y_1 and y_2.

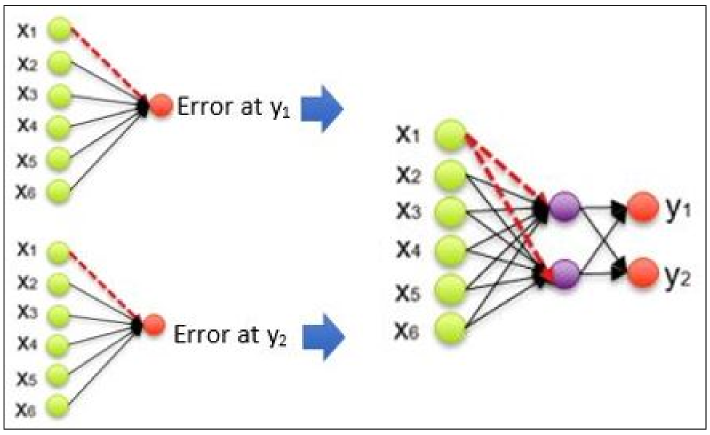


Figure 3.7: (Left) Two neural networks trained on the prediction errors of the output states; (Right) The likely location of the changed weights narrowed to the two dotted weights in red.

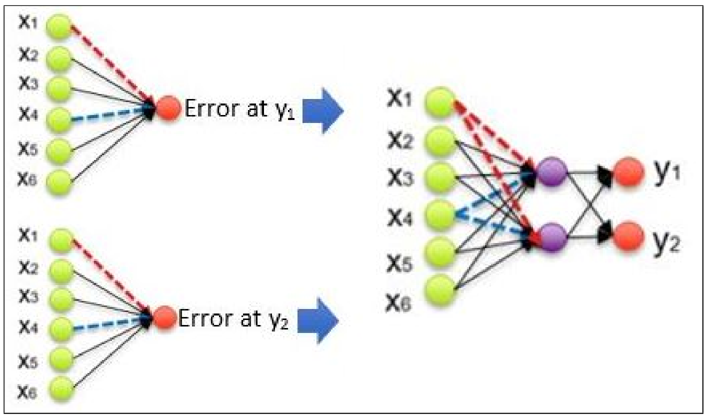


Figure 3.8: Likely location of the changed weights narrowed down to the four dotted weights in red and blue.

In both these examples, observing patterns in values of the weights in the newly trained neural networks helps conclude that changed weight(s) only lie in W_1 and therefore the weights in W_2 remain unchanged. This allows for the unchanged weights in W_2 to be used for estimating the values of the neurons 'a' and 'b' in the hidden layer as shown in figure 3.9. In this figure, it is seen that the weights in W_2 remain unchanged (NN_{13} = 6, NN_{14} =2, NN_{15} = 3 and NN_{16} = 5), and the output of the faulty system at time-step t = 5 for the output states y_1 and y_2 is known from the measurement data to be 60 and 45, respectively. The value of the hidden neurons 'a' and 'b' at time instance t = 5 is therefore calculated by solving the pair of linear equations given in equation 3.13. This is repeated to calculate the value of hidden layer neurons for all time-steps.

$$\begin{aligned} 3a + 5b &= 45 \\ 6a + 2b &= 60 \end{aligned} \tag{3.13}$$

The limitation of calculating the values of hidden layer neurons in this manner is that the accuracy of the values calculated is highest when the number of neurons in hidden layer is equal to the number of neurons in the output layer. This is because the number of unknowns (values of hidden layer neurons) to be estimated is equal to the number of known equations that can be formed (deterministic equations). An example of this limitation is shown in figure 3.10 where the values of three neurons a, b and c in the hidden layer are to be estimated from the two known equations (equation 3.14). The accuracy of the values obtained for neurons a, b and c is expected to be poor due to the under-deterministic nature of the equations

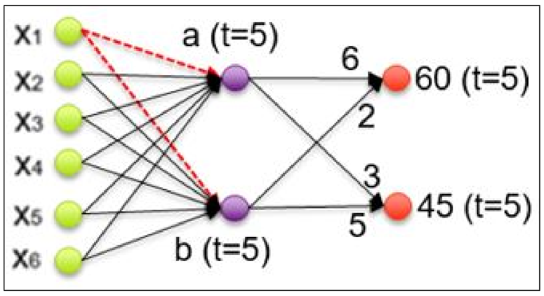


Figure 3.9: Calculating the values of the neurons ’a’ and ’b’ in the hidden layer at time step t = 5.

(3 unknown variables and 2 equations).

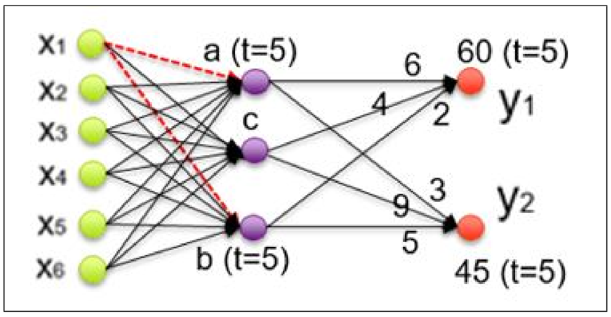


Figure 3.10: Calculating the value of the three unknown hidden layer neurons a, b and c.

$$\begin{aligned} 3a + 5b + 9c &= 45 \\ 6a + 2b + 4c &= 60 \end{aligned} \tag{3.14}$$

Following the estimation of values of neurons in the hidden layer, the reconstruction of neural network’s weights ($NN_i^{'}$) in W_1 can be done by formulating a sparse signal recovery problem between the neurons of the input and hidden layers and then solving it by applying the sparse Bayesian learning algorithm for the linear case (algorithm 1). In the formulated SSR problem, y is a vector of dimension M×1 that represents the estimated values of the hidden neurons ’a’ or ’b’ over M measurements. The dictionary matrix Φ of dimension M×6 represents the 6 input states of the system (x_1 to x_6) across the M measurements. Finally, the vector x of dimension 6×1 contains the required reconstructed weights $NN_i^{'}$. Following the reconstruction of the weights, the fault isolation is then performed as per algorithm 2.

– **Situation 2: True location of changed weight(s) lies between the hidden and output layers (W_2) only**
Consider that a fault in the linear system results in changing of one of the neural network’s weights that lies between the hidden and output layers (W_2). Assume this weight to be NN_{13}, which after the occurrence of the fault changes its value to $NN_{13}^{'}$. The prediction errors at both the output states as a result of this fault is shown in figure 3.11 (left). From this figure it can be seen that the prediction error for output state y_2 is negligible even after the occurrence of the fault. However, this is not the case with the prediction error on output state y_1. This type of pattern implies that one of the weights in W_2 has changed. Moreover, it can only be either one of the weights connecting the neurons of the hidden layer to the first neuron of the output layer (output state y_1), as indicated by the pink dotted weights (NN_{13} and NN_{14}) in figure 3.11(right).
Additionally, training a new neural network (figure 3.11 (middle)) on the prediction error of output state y_1 reveals a distinctive pattern in values of the trained weights,

as listed in table 3.1 (situation 2 example 1). All values of the weights of the neural network are non-zero and significant in their magnitudes. Repeated simulation shows that this pattern re-occurs whenever the true location of the changed-weight belongs to the set of weights in W_2 of the neural network modelling the linear system.

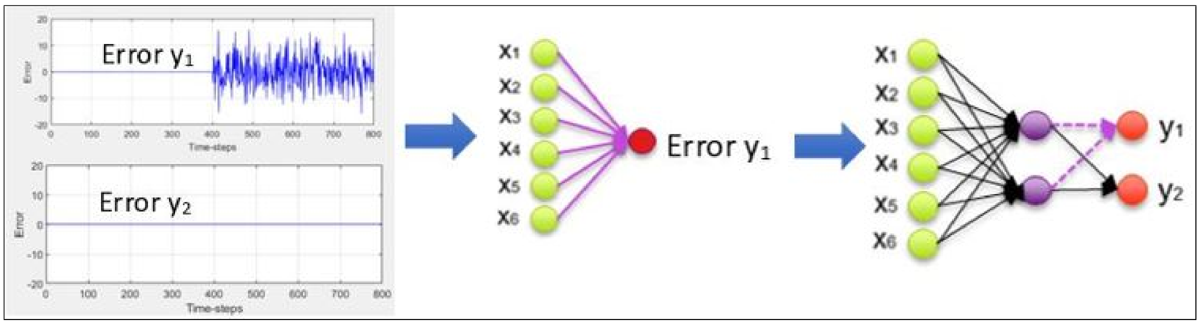


Figure 3.11: (Left) Prediction errors at output states y_1 and y_2. (Middle) Neural network trained on prediction error at output state y_1. (Right) Likely weights that changed in the neural network after the occurrence of fault.

With the intuition that the changed weight only lies in W_2 and that all weights in W_1 remain unchanged, the value of the hidden layer's neurons 'a' and 'b' can be easily calculated by performing a single feed-forward step. Following the estimation of values of the neurons in the hidden layer, the reconstruction of the neural network's weights ($NN_i^{'}$) in W_2 can be done by formulating a sparse signal recovery problem between the neurons of the hidden and outer layers and then solving it by applying the sparse Bayesian learning algorithm for the linear case (algorithm 1). In the formulated SSR problem, y is a vector of dimension M×1 that represents the output state of the system (y_1) across the M measurements. The dictionary matrix Φ of dimension M×2 represents both hidden layer neurons (a and b) across the M measurements. Finally, the vector x of dimension 2×1 contains the required reconstructed weights $NN_i^{'}$ in W_2. Following the reconstruction of the weights, the fault isolation is then performed as per algorithm 2.

– **Situation 3: True location of changed weight(s) lies in both W_1 and W_2 layers.** Consider that a fault in the linear system results in changing of multiple neural network weights that belong to the set of weights in both W_1 and W_2. Assume these weights to be NN_1 (in W_1) and NN_{16} (in W_2), which after the occurrence of the fault change their values to $NN_1^{'}$ and $NN_{16}^{'}$, respectively. The prediction errors at both the output states as a result of this fault are shown in figure 3.12 (left). From this figure , it is safe to rule out the possibility that the changed weights lie only in W_2 (situation 2). However, no comments can be made yet about whether the changed weights lie only in W_1 (situation 1) or both W_1 and W_2 (situation 3). In order to differentiate between these two, a distinctive pattern must be identified in the values of the weights of the neural networks that have been trained on the prediction errors of the output states.

The identified weights following the training of new neural networks on the prediction errors (figure 3.12 (middle)) are listed in table 3.1 (situation 3 example 1). The weights of the neural network that was trained on the prediction error of output state y_1 replicates a pattern similar to the one observed in situation 1 i.e. one of the weights (here the first weight) has a large magnitude while the other weights have a low or nearly zero value. This pattern indicates that at-least one of the changed weights lie in W_1 and it's location can be narrowed down to any one of the weights going from the first input neuron to the rest of the neurons in the hidden layer i.e. NN_1 or NN_7. The value of the weights of the neural network that was trained on

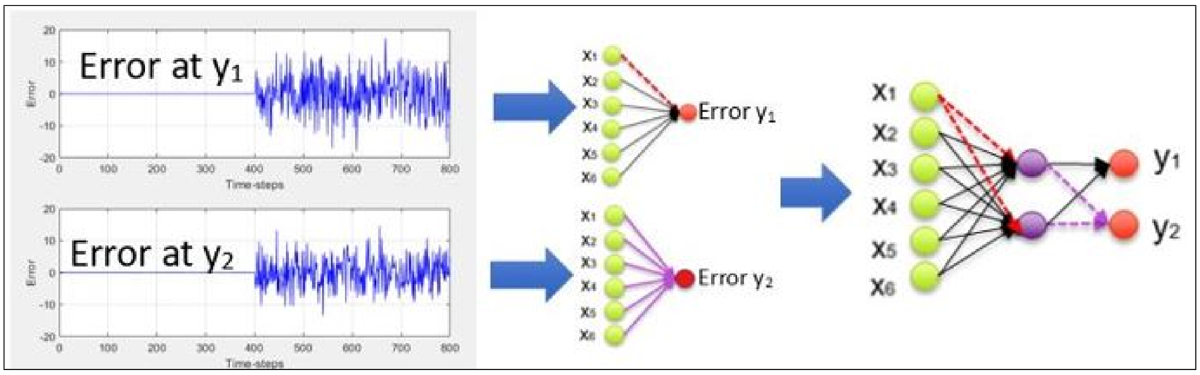


Figure 3.12: (Left) Prediction errors at output states y_1 and y_2. (Middle) Neural network trained on prediction error of output state y_1 and y_2. (Right) Likely weights that changed in the neural network after the occurrence of fault.

the prediction error of the output state y_2 replicates a pattern similar to the one observed in situation 2, i.e. all the weights are non-zero and of certain magnitude in value. This pattern indicates that at-least one of the changed weight(s) lies in W_2 and it's location can be narrowed down to any one of the weights going from the second neuron of the hidden layer (because the pattern appeared in the weights of the neural network that was trained on prediction error of output state y_2) to the rest of the neurons in the output layer, i.e. NN_{15} or NN_{16}. As a result, the possible locations of the changed weights are narrowed down to the red and pink dotted weights shown in figure in 3.12 (right).

Following the intuition on the possible locations of the changed weights, the next step is to treat the red dotted weights (NN_1 and NN_7) in W_1 as variables and carry out a single feed-forward step to obtain the values of neurons in the hidden layer as a function of these variables. This is followed by the reconstruction of the neural network's weights ($NN_i^{'}$) in W_2 by formulating a sparse signal recovery problem between the neurons of the hidden and output layers that is solved by the sparse Bayesian learning algorithm (algorithm 1). Now that the weights in W_2 have been reconstructed, the numeric values of the neurons in the hidden layer can be estimated by solving pairs of linear equations as was done in situation 1. This is followed by the reconstruction of the neural network's weights ($NN_i^{'}$) in W_1 by formulating another sparse signal recovery problem, but this time between the neurons of the input and hidden layers. This is then solved by applying the sparse Bayesian learning algorithm (algorithm 1). Following the reconstruction of weights in both W_1 and W_2, the fault isolation is then performed as per algorithm 2.

- **Case 3: Fault isolation and identification for nonlinear system modelled by neural networks having a hidden layer with a nonlinear activation function.**
  Consider a generic nonlinear system that has six input states x_1 to x_6 and two output states y_1, y_2 respectively. Let the normal operating dynamics of the nonlinear system be described by equation 3.15, where θ_1 to θ_{12} are fixed coefficients of the input states and γ_1, γ_2 are the measurement noise that is assumed to be Gaussian.

$$\begin{aligned} y_1 &= \theta_1 * x_1^2 + \theta_2 * x_2 + \theta_3 * x_3 + \theta_4 * x_4^2 + \theta_5 * x_5 + \theta_6 * x_6 + \gamma_1 \\ y_2 &= \theta_7 * x_1 + \theta_8 * x_2^2 + \theta_9 * x_3 + \theta_{10} * x_4 + \theta_{11} * x_5 + \theta_{12} * x_6^2 + \gamma_2 \end{aligned} \tag{3.15}$$

  Let the system be modelled by the same neural network architecture used in the previous case (figure 3.5). Once trained, the system dynamics can be represented in terms of the weights of the neural network as per equation 3.16, or in the matrix form as given in equation 3.17.

$$[y_1 \quad y_2] = f\left([x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6] \times \begin{bmatrix} NN_1 & NN_7 \\ NN_2 & NN_8 \\ NN_3 & NN_9 \\ NN_4 & NN_{10} \\ NN_5 & NN_{11} \\ NN_6 & NN_{12} \end{bmatrix} \right) \times \begin{bmatrix} NN_{13} & NN_{15} \\ NN_{14} & NN_{16} \end{bmatrix} \tag{3.16}$$

$$Y = f(X \times W_1) \times W_2 \tag{3.17}$$

$$f(Z) = sigmoid(Z) = \frac{1}{1+e^{-Z}} \tag{3.18}$$

Here f is the sigmoid activation function (nonlinear) on the hidden layer given by equation 3.18. W_1 and W_2 represent the set of weights connecting the input layer to the hidden layer and the hidden layer to the output layer of the neural network, respectively.
Due to the presence of a hidden layer in the neural network modelling the nonlinear system (figure 3.5), the weight reconstruction process for all weights in W_1 and W_2 has to be carried out in two stages (similar to the previous case). As mentioned earlier, this reconstruction is not direct and requires estimating the values of the neurons in the hidden layer. This is done by training new neural networks on the prediction errors at the output states in order to locate the weights in the neural network that have most likely changed their value after the occurrence of the fault in the system. The architecture of the neural networks that are trained on the prediction errors of output states differ from the ones in the previous case and is shown in figure 3.13 (right). The input layer consists of 6 neurons which correspond to each input state (x_1 to x_6) of the system. The hidden layer consists of two neurons for which the activation function is a nonlinear one (sigmoid function). The output layer consists of one neuron that represents the prediction errors for the output states (y_1 and y_2) of the system.

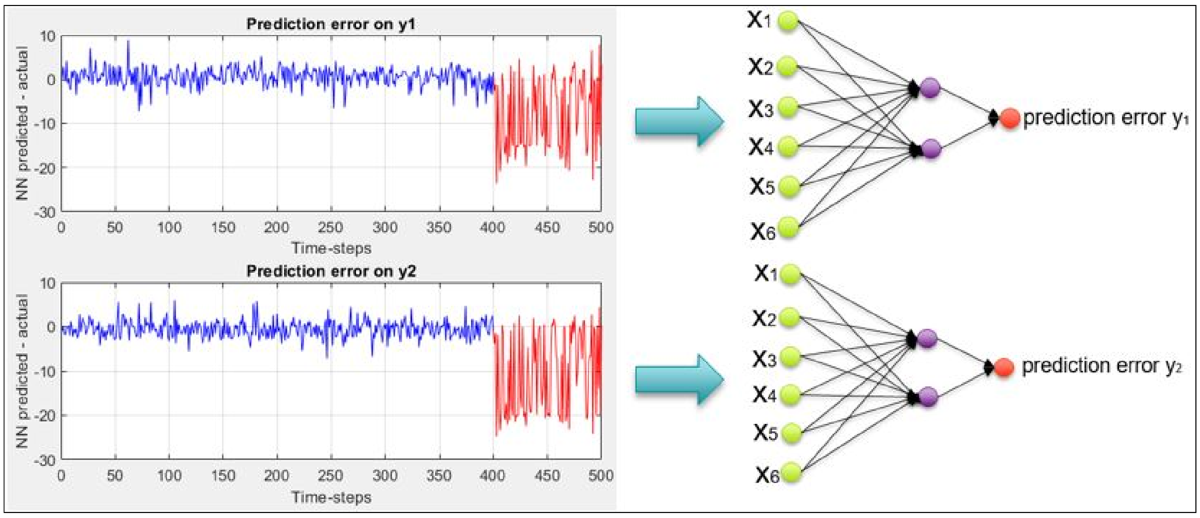


Figure 3.13: Case 3: Architecture of the neural networks (right) that are trained on the prediction errors at the output states (left).

Practical experience shows that distinctive patterns in values of weights of the newly trained neural networks are observed depending on true location(s) of the changed-weight(s) in the neural network modelling the nonlinear system. Recognizing these patterns helps narrow down the possible locations of the changed weights from many to only a few in number. These distinctive patterns are described below:

– **Situation 1: True location of changed weight(s) lies between the hidden and output layers (W_2) only**
Consider that a fault in the non-linear system results in changing of one of the neural network's weights that lies between the hidden and output layers (W_2). Assume this weight to be NN_{13}, which after the occurrence of the fault changes its value to NN_{13}'. The prediction errors at both the output states as a result of this fault are shown in figure 3.14 (left). The prediction error for output state y_2 can be seen to be negligible after the occurrence of the fault. However, this is not the case with the prediction error on output state y_1. Training a new neural network on the prediction error of output state y_1 (figure 3.14 (middle)) reveals a distinctive pattern in the values of the trained weights, as listed in table 3.2 (situation 1 example 1). It can be seen that all pairs of weights in W_1 going from any one neuron in the input layer to the neurons in the hidden layer (NN_1&NN_7, NN_2&NN_8, NN_3&NN_9, NN_4&NN_{10}, NN_5&NN_{11} and NN_6&NN_{12}) are nearly equal in magnitude and same in sign to one another. In addition to this, both weights in W_2 vary significantly from each other in magnitude and are same in sign. The combination of these patterns indicates that a changed-weight is located in W_2 of the neural network modelling the system. Furthermore, its location can be narrowed down to either one of the weights connecting the hidden layer's neurons to the first neuron of the output layer (output state y_1), as indicated by the red dotted weights (NN_{13} and NN_{14}) in figure 3.14(right).

Note: *the prediction errors on the output states in figure 3.14 (left) in itself is an indicator that the changed-weight is located in the set of weights connecting the hidden and outer layer, and therefore intuitively could be either NN_{13} or NN_{14}.*

Similarly, when multiple weights in W_2 change, for example NN_{13} and NN_{15}, patterns observed in the weights of both newly trained neural networks are listed in table 3.2 (situation 1 example 2). A similar pattern is seen in the weights of both neural networks, i.e. all weight-pairs in W_1 are nearly equal in magnitude and same in sign while the weights in W_2 are of same sign but vary significantly from each other in magnitude. Based on these patterns an educated guess can be made that all weights other than the red dotted weights ($NN_{13}, NN_{14}, NN_{15}$ and NN_{16}) in figure 3.15 (right) remain unchanged.

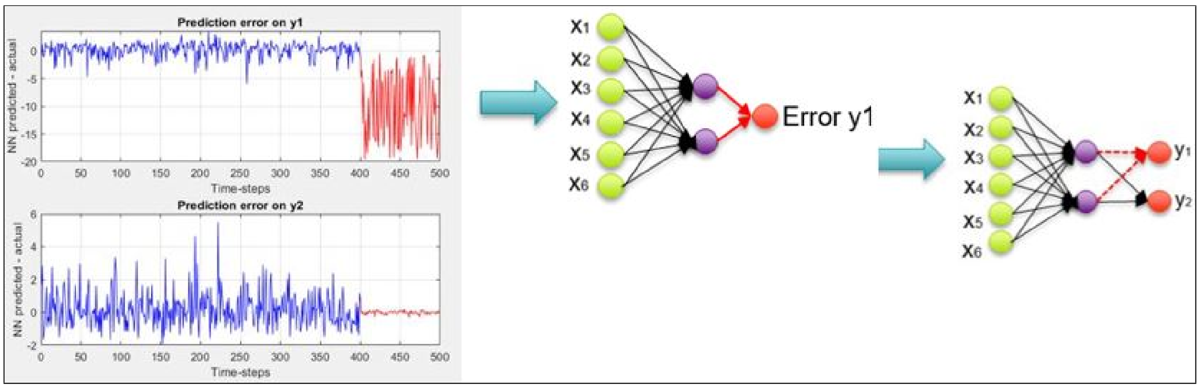


Figure 3.14: (Left) Prediction error in the output state y_1. (Middle) Neural network trained on the prediction error. (Right) Red dotted weights represent the likely weights that changed after fault.

Neural Network #	Weight Layer W1 or W2	Situation 1: Changed weight located b/w hidden and output layer only (W2 only)				Situation 2: Changed weight located in both W1 and W2 layers				Situation 3: Changed weight located b/w hidden and output layer only (W1 only)	
		Example 1		Example 2		Example 1		Example 2		Example 1	
Neural Network 1 trained on prediction error at output state 1	W1	-0.2438	-0.1265	-0.2438	-0.1265	4.6780	-0.3209	4.3599	0.6604	-0.4079	8.0140
		0.2183	0.1123	0.2183	0.1123	-1.5386	-1.4567	-1.5959	-1.6202	0.5091	0.5702
		1.5311	1.5851	1.5311	1.5851	0.4224	0.4143	0.5491	0.5485	1.6024	1.6677
		0.2348	0.1082	0.2348	0.1082	-0.7519	-0.7357	-0.7594	-0.7815	-0.0788	-0.1003
		-0.3378	-0.4206	-0.3378	-0.4206	0.1436	-0.1324	0.3553	0.3829	-0.5727	-0.6016
		-0.8694	-0.9631	-0.8694	-0.9631	0.5254	0.5231	0.3868	0.3284	-1.3139	-1.4031
	W2	-8.0663		-12.3500		-13.4682		-6.5620		9.1605	
		-11.9399		-7.6955		-6.5276		-8.3888		-9.1481	
Neural Network 2 trained on prediction error at output state 2	W1	N/A	N/A	-0.2470	-0.1397	-1.9714	1.6995	-0.6336	4.4693	-6.2288	0.3862
		N/A	N/A	0.1413	0.2021	0.4798	1.0536	-1.5531	-1.6243	-0.3243	-0.5400
		N/A	N/A	1.5043	1.6227	-1.0924	0.9339	0.5471	0.5642	-1.3339	-1.6707
		N/A	N/A	0.3316	0.2448	1.1345	0.1172	-0.7532	-0.7741	0.0194	0.0456
		N/A	N/A	-0.5530	-0.4051	0.5360	0.3440	0.3619	0.3737	0.5258	0.5838
		N/A	N/A	-0.8980	-0.9399	-0.1557	-1.2288	0.2951	0.3046	1.0656	1.3555
	W2	N/A		-10.0092		4.6565		-7.8626		4.9567	
		N/A		-9.9364		1.9511		-7.1988		-4.9677	

Table 3.2: Nonlinear systems: Distinctive patterns in the values of the weights of the new neural networks trained on the prediction errors of system's output states y_1 and y_2.

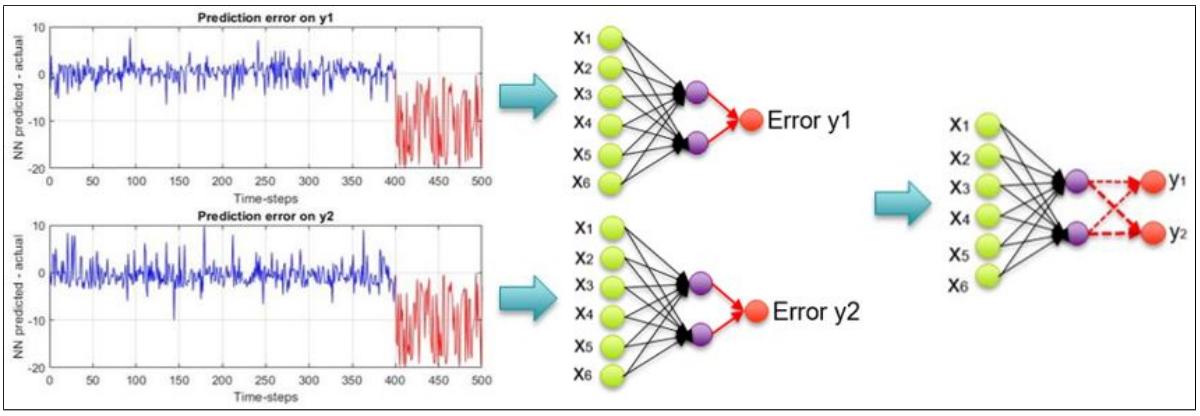


Figure 3.15: (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Red dotted weights represent the likely weights that changed after fault.

With the intuition that the changed-weight(s) in the neural network modelling the system only lies in W_2 and that all weights in W_1 remain unchanged, the value of the hidden layer's neurons can be easily calculated by performing a single feed-forward step, while keeping in mind the sigmoid activation function on the hidden layer. Following the estimation of the values of neurons in the hidden layer, the reconstruction of the neural network's weights ($NN_i^{'}$) in W_2 can be done by formulating a sparse signal recovery problem between the neurons of the hidden and output layers, and then solving it by applying the sparse Bayesian learning algorithm for the linear case (algorithm 1). In the formulated SSR problem, y is a vector of dimension M×1 that represents the output state of the system (y_1 or y_2) across the M measurements. The dictionary matrix Φ of dimension M×2 represents both hidden layer neurons across the M measurements. Finally, the vector x of dimension 2×1 contains the required reconstructed weights $NN_i^{'}$ in W_2. Following the reconstruction of the weights, the fault isolation is then performed as per algorithm 2.

– **Situation 2: True location of changed weight(s) lies in both the W_1 and W_2 layers.**
Consider that a fault in the nonlinear system results in changing of multiple trained neural network's weights that belong to the set of weights in both W_1 and W_2. Assume these weights to be NN_1 (in W_1) and NN_{13} (in W_2), which after the occurrence of the fault change their values to $NN_1^{'}$ and $NN_{13}^{'}$, respectively. The prediction errors at both output states as a result of this fault can be seen in figure 3.16 (left). Training new neural networks on the prediction errors of the output states (figure 3.16 (middle)) reveals a distinctive pattern in the values of the trained weights, as listed in table 3.2 (situation 2 example 1). In the first trained neural network, for the weights in W_1, with the exception of the first pair of weights (NN_1&NN_7) the other weight-pairs (NN_2&NN_8, NN_3&NN_9, NN_4&NN_{10}, NN_5&NN_{11} and NN_6&NN_{12}) are nearly equal in magnitude and same in sign. Additionally, the weights in W_2 vary significantly from each other in magnitude but are same in sign. In the second trained neural network, for the weights in W_1, all pairs of weights are significantly different from each other in magnitude.
From the combination of these patterns it can be inferred that one of the changed-weights is located in W_2 of the neural network modelling the nonlinear system. Furthermore, its location can be narrowed down to either one of the weights connecting the hidden layer's neurons to the first neuron of the output layer (output state y_1) i.e., NN_{13} or NN_{14}, as shown by the red dotted weights in figure 3.16 (right). In addition to this, the mismatch in magnitudes of the first pair of weights in W_1 of the first neural network implies a second changed-weight is located in W_1 of the neural network modelling the nonlinear system. The location of this weight can be

narrowed down to either one of the weights connecting the first neuron of the input layer to all the neurons in the hidden layer, i.e. NN_1 or NN_7, as shown by the blue dotted weights in figure 3.16 (right).

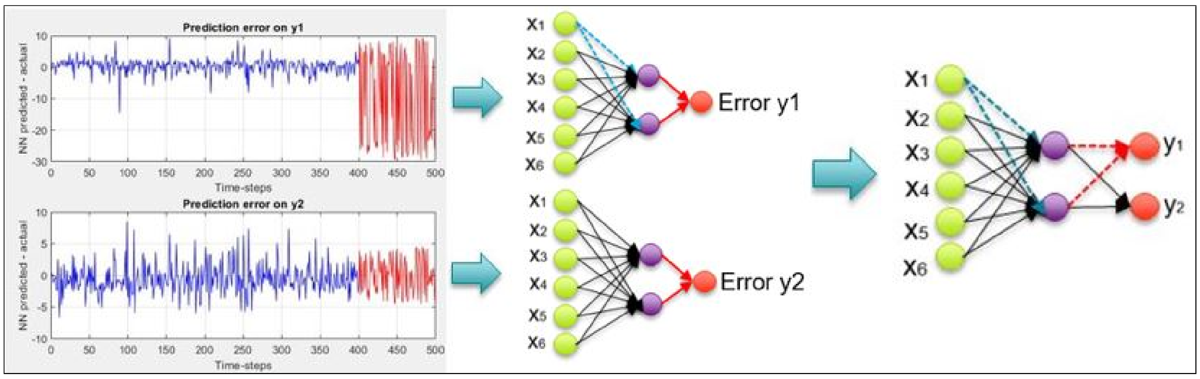


Figure 3.16: (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Red dotted weights represent the likely weights that changed in the neural network after fault.

Consider an extension where in addition to NN_1 and NN_{13}, the NN_{15} weight in W_2 also changes after the occurrence of fault in the system. Training new neural networks on the prediction errors (figure 3.17 (middle)) of the output states results in the identified weights as listed in table 3.2 (situation 2 example 2). The pairs of weights in W_1 of the first neural network show a similar pattern as before, i.e. all pairs of weights with the exception of the first pair are nearly equal in magnitude and sign. Moreover, the weights in W_2 layer for the first neural network are unequal in magnitude and same in sign. From the combination of these patterns it can be inferred that one of the changed-weights is located in W_2 of the neural network modelling the nonlinear system. Furthermore, its location can be narrowed down to either one of the weights connecting the hidden layer's neurons to the first neuron of the output layer (output state y_1)i.e., NN_{13} or NN_{14}, as shown in the red in figure 3.17 (right). The weights in W_1 and W_2 of the second neural network shows the same pattern as the weights in W_1 and W_2 of the first neural network, which implies that a second changed weight is located in the W_2 of the neural network modelling the nonlinear system. This weight's location can be narrowed down to either one of the weights connecting the hidden layer's neurons to the second neuron of the output layer (output state y_1), i.e. NN_{15} or NN_{16}, as shown in red in figure 3.17 (right). Lastly, the mismatch in magnitudes of the first pair of weights in W_1 for both the neural networks implies that a third changed weight exists and is located in W_1 of the neural network modelling the system. Its location can be narrowed down to one of the weights going from the first neuron of the input layer to all the neurons in the hidden layer, i.e. W_1 or W_7, as shown by the blue dotted weights in figure 3.17 (right).

Following the intuition on the possible locations of the changed weights, the next step is to treat the dotted weights in W_1 as variables and carry out a single feed-forward step in order to obtain the values of neurons in the hidden layer as a function of these variables. This is followed by the reconstruction of the neural network's weights in W_2 by formulating a sparse signal recovery problem between the neurons of the hidden and output layers that is solved by the sparse Bayesian learning algorithm (algorithm 1). Now that the weights in the W_2 have been reconstructed, the numeric values of the neurons in the hidden layer can be estimated by solving pairs of linear equations. Finally, the reconstruction of the neural network's weights in W_1 can be done by formulating a sparse signal recovery problem between the neurons of the input and hidden layers that can be solved by applying

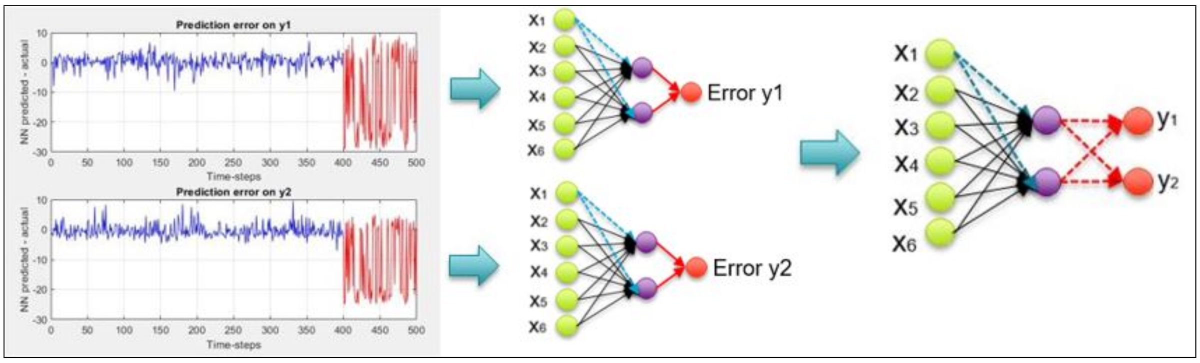


Figure 3.17: (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Red and blue dotted weights represent the likely weights that changed in the neural network after fault.

the sparse Bayesian learning algorithm for the nonlinear case, as shown in algorithm 3. The only difference between the two versions (nonlinear and linear) of the SBL algorithm is to account for the nonlinear activation function applied on the hidden layer of the neural network modelling the nonlinear system. Following the reconstruction of the weights, fault isolation is then performed as per algorithm 2.

Algorithm 3 Re-weighted L_1 type sparse Bayesian learning algorithm for reconstruction of the neural network weights (fault identification) in the nonlinear case

```
1: Fault Identification by weight reconstruction:
2: Initialize $\lambda = 1$ and $w_i^1$ as unit vector;
3: Assign B to be an identity matrix of dimension 6×6; B = eye(6);
4: for $k = 1, \ldots\ldots, k_{max}$ do
5:     $x^{k+1} = argmin_x(\|y - f(\Phi x)\|_2^2 + \lambda \sum_{i=1}^{6} \|w_i^k .* B_{i,:}x\|_1)$ ;
6:     $\gamma_i^{k+1} = \left|\frac{B_{i,:}x^{k+1}}{w_i^k}\right|$ ;
7:     $\tau^{k+1} = diag(\gamma_i^{k+1})$ ;
8:     $C^{k+1} = (B^T(\tau^{k+1})^{-1}B + H)^{-1}$ ;
9:     $\alpha_i^{k+1} = -\frac{B_{i,:}C^{k+1}B_{i,:}^T}{(\gamma_i^{k+1})^2} + \frac{1}{\gamma_i^{k+1}}$ ;
10:    $w_i^{k+1} = \sqrt{\alpha_i^{k+1}}$ ;
11:    if $\gamma_i^{k+1}$ has converged or a stopping criterion has been reached then
12:        break;
13:    end if
14: end for
15:
16:
17: Where $B_{i,:}$ stands for the $i^{th}$ row of the matrix B
18: Where $H$ stands for the hessian matrix
19: Where $.*$ represents dot product
```

– **Situation 3: True location of changed weight(s) lies between the input and hidden layers (W_1) only**
Consider that a fault in the nonlinear system results in changing of one of the neural network's weights that lies between the input and hidden layers (W_1). Assume this weight to be NN_1, which after the occurrence of the fault changes its value to NN_1'. The prediction errors at the output states as a result of this fault can be seen in figure 3.18 (left). Training two new neural networks on the prediction errors (figure 3.18 (middle)) for output states y_1 and y_2 reveals a distinctive pattern

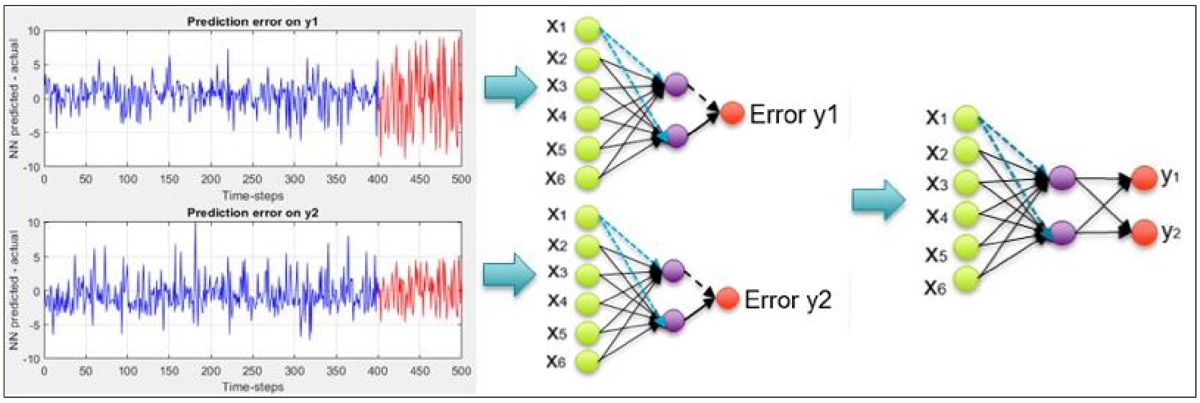


Figure 3.18: (Left) Prediction errors in the output states y_1 and y_2. (Middle) Neural network trained on the prediction errors. (Right) Blue dotted weights represent the likely weights that changed in the neural network after fault.

in the values of the trained weights as listed in table 3.2 (situation 3 example 1). In both the neural networks, for the weights in W_1, with the exception of the first pair of weights (NN_1 & NN_7) the other weight-pairs (NN_2&NN_8, NN_3&NN_9, NN_4&NN_{10}, NN_5&NN_{11} and NN_6&NN_{12}) are nearly equal in magnitude and same in sign to each other. Meanwhile looking at the value of the weights in W_2, for both the neural networks it can be seen that both weights are nearly equal in magnitude and opposite to each other in sign. From the combination of these patterns it can be inferred that one of the changed-weights is located in W_2 of the neural network modelling the nonlinear system. Furthermore, its location can be narrowed down to either one of the weights connecting the hidden layer's neurons to the first neuron of the output layer (output state y_1), i.e. NN_{13} or NN_{14}, as shown in the red in figure 3.17 (right). From the combination of the observed patterns in both neural networks it can be inferred that the changed weight is located in W_1 of the neural network modelling the system. Furthermore, its location can be narrowed to either one of the weights connecting the first neuron in the input layer to both neurons in the hidden layer, i.e. W_1 or W_7 weights shown in blue in figure 3.18 (right).

Following the intuition that the changed weights are located only in W_1 of the neural network modelling the nonlinear system and that the weights in W_2 layer of this neural network remain unchanged, the value of neurons in the hidden layer can be calculated by solving the required linear equations. The reconstruction of the neural network's weights in W_1 can then be carried out by formulating a sparse signal recovery problem between the neurons of the input and hidden layer. This SSR problem can then be solved by applying the sparse Bayesian learning algorithm for the nonlinear case (algorithm 3).

Note: *that the pattern of the value of weights in W_2 of both neural networks is what differentiates this situation (situation 3) from previous situation (situation 2), where the weights in W_2 of both the neural networks are same in sign and had large difference in their magnitudes.*

3.3. Discussion

Some final points worth mentioning about the process of fault detection, isolation and identification described above are:

- The fault detection task can be done online.
- The weight reconstruction (fault identification) task involves computationally expensive calculations which although can be done online, are not possible to implement during real time operation.

- Although there can be an occurrence of more than one fault at any given time instance, while the data corresponding to the faulty system is being recorded for the reconstruction of weights, it is assumed that no other faults occur in the system during this brief period.

In this chapter, the methodology for carrying out fault detection and fault identification and isolation via reconstruction of neural network weights by applying sparse Bayesian learning algorithm was discussed. For a linear system modelled by a neural network that does not have a hidden layer, the fault diagnosis process is straightforward. However for the linear or nonlinear systems modelled by a neural network having a hidden layer, in order to reconstruct the weights, first the values of the neurons of the hidden layers have to be estimated. This estimation requires having some insights into the possible locations of the weight(s) that have changed after the occurrence of the fault. It is shown that this insight can be gained by identifying distinctive patterns in the values of the weights of the neural networks that have been newly trained on the prediction errors of the output states. The distinctive patterns help identify whether the location of the changed-weight(s) lie either in W_1, W_2 or in both. This insight is then used to estimate the values of the neurons in the hidden layer. Finally, the reconstruction of the weights across the whole neural network that models the system is carried out.

4 Experiments

In the previous chapter, fault diagnosis algorithms for carrying out fault detection and fault identification & isolation of a generic linear and nonlinear system were proposed and their implementations were discussed. The goal of this chapter is to apply the proposed algorithms to (hypothetical) linear and non-linear systems modelled by neural networks, to demonstrate their suitability for carrying out a fault diagnosis task.

4.1. Experiment 1: Linear system modelled by neural network without hidden layer

Consider a linear system that has six input states x_1 to x_6 and two output states y_1 y_2. Let the normal operating dynamics of the linear system be described by equation 4.1, where γ_1 and γ_2 are the measurement noise at output states y_1 and y_2, respectively, and are assumed to be Gaussian.

$$\begin{aligned} y_1 &= 0.5x_1 + 2x_2 - 1.1x_3 + 0.95x_4 + 0.5x_5 - 2x_6 + \gamma_1 \\ y_2 &= 3x_1 + 2.5x_2 + x_3 + 0.5x_4 - 0.5x_5 + 1.5x_6 + \gamma_2 \end{aligned} \tag{4.1}$$

Step 1: Data generation from the linear system
For the purpose of this study, synthetic data corresponding to the normal operating conditions of the system is generated with the help of true equations of the system given in equation 4.1. All input states (x_1 to x_6) are Gaussian $\mathcal{N}(0,1)$ with mean = 0 and variance = 1. The measurement noises γ_1 γ_2 are also assumed to be Gaussian $\mathcal{N}(0,0.01)$ with mean = 0 and variance = 0.01. This ensures the presence of noise in the output signals whilst maintaining a good signal to noise ratio. Figure 4.1 shows the generated training data set for the input (left) and output (right) states of the system.

Step 2: Modelling linear system with neural network
The neural network architecture chosen to model the linear system is shown in figure 3.3. The neural network's input layer has six neurons, each corresponding to one of the input states of the system (x_1 to x_6). The output layer has two neurons, each corresponding to one of the output states (y_1 & y_2) of the system. The next step is to train this neural network on the training data set. Once trained, the neural network is cross-validated on the training and test data sets as shown in figure 4.2. Here the red line and the blue stars represent the predicted and target outputs, respectively. The mean-square errors (MSE) obtained on the training and test data sets are listed in table 4.1. As expected, the neural network accurately models the linear system. Moreover, it is able to generalize to data it has not been trained on, as is evident from its modelling accuracy on the test data set. The trained weights of this network are given in table 4.3 (second column). Having trained the neural network, the linear system can now be represented as a function of the neural network's weights, given by equation 3.5.

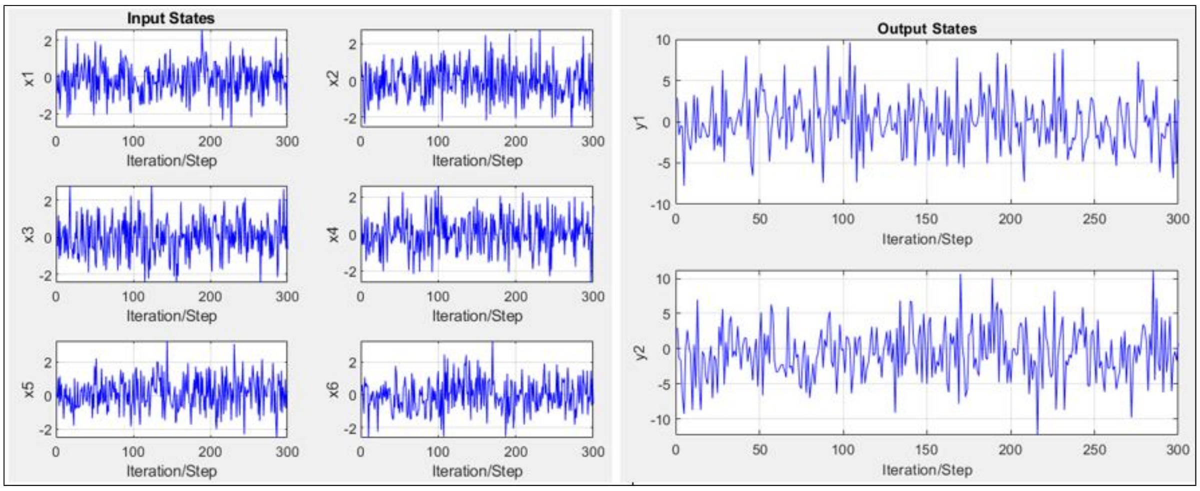


Figure 4.1: Experiment 1: Training data set corresponding to the linear system's six inputs (left) and two output states (right) under normal operating conditions.

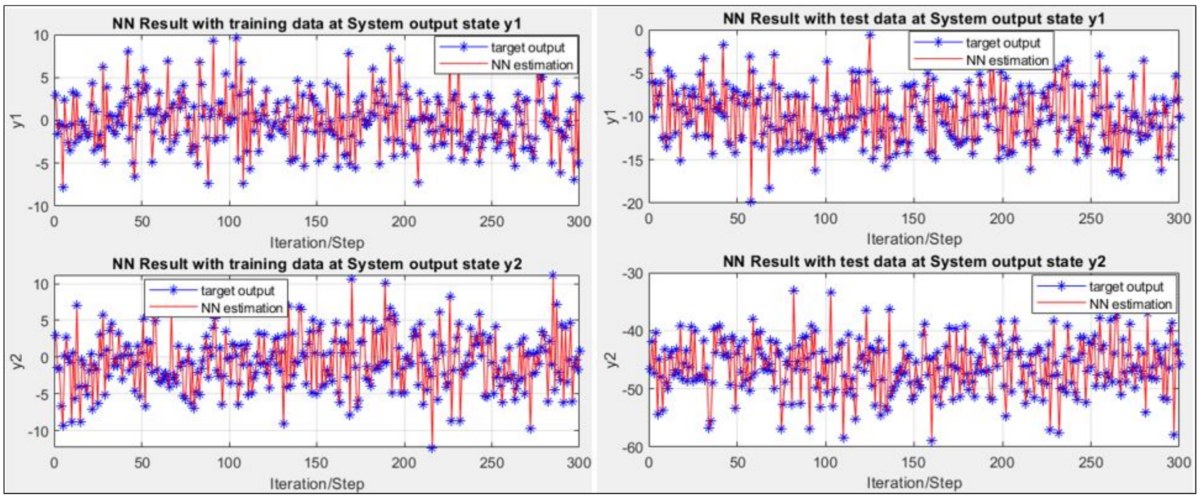


Figure 4.2: Experiment 1: Cross-validation of the trained neural on training (left) and test data sets (right).

Output State	MSE on training data set	MSE on test data set
y1	0.0094	0.0109
y2	0.0086	0.0136

Table 4.1: Experiment 1: Mean-squared error on the training and test data sets.

The low MSE obtained on the training data set indicates that on average, the magnitude of the modelling errors i.e. residuals r_i $(= y_t - y_m)$ for each data point 'i' in the training data set are low. Training a model on the current input-residual pair data sets $((u_i, r_i)_{i=1}^{N})$ will result in an adaptive threshold model whose output y_e and variance v are very low. This in turn implies that the upper and lower thresholds calculated by this model as per equations 3.3 and 3.4 respectively, will also be low in their mean and variance. The low variance of the thresholds essentially means that they will behave similar to static thresholds. Hence, for this experiment, a static threshold was implemented instead.
Ignoring the contribution of the y_e term in equations 3.3 and 3.4, the static upper and lower thresholds are calculated as per equations 4.2 and 4.3, respectively.

$$T_u = t_\beta v \tag{4.2}$$

$$T_l = -t_\beta v \tag{4.3}$$

Although, originally, v is the standard deviation of y_e, due to the absence of the model, v is assigned the standard deviation of the residuals. This was found to be 0.0966 and 0.0928 at the system's output states y_1 and y_2, respectively. For a 95% confidence level, t_β was assigned a value of 1.96. The upper and lower thresholds were calculated to be ± 0.1893 and ± 0.1819 for the system's output states y_1 and y_2, respectively.

Step 3: Occurrence of a fault in system

After the neural network has been trained offline, it is ready for its online implementation alongside the linear system, where it will monitor and detect the occurrence of faults and perform fault identification and isolation tasks after a fault has been detected in the system. At time-step t = 301 during it's operation, a fault in the system occurs, as a result of which the weights NN_1 & NN_9 change their values to $NN_1^{'}$ & $NN_9^{'}$, as per equation 4.4. From time-step 301 onward, the faulty system generates data as per equation 4.5. Figure 4.3 shows the effect of the fault on the system's output states y_1 (top) and y_2 (bottom). Here the blue and red plots represents the neural network's predicted output and the system's output (after fault), respectively.

$$\begin{aligned} NN_1^{'} &= NN_1 + 3 \\ NN_9^{'} &= NN_9 - 5 \end{aligned} \tag{4.4}$$

$$\begin{aligned} y_1 &= NN_1^{'} * x_1 + NN_2 * x_2 + NN_3 * x_3 + NN_4 * x_4 + NN_5 * x_5 + NN_6 * x_6 + \gamma_1 \\ y_2 &= NN_7 * x_1 + NN_8 * x_2 + NN_9^{'} * x_3 + NN_{10} * x_4 + NN_{11} * x_5 + NN_{12} * x_6 + \gamma_2 \end{aligned} \tag{4.5}$$

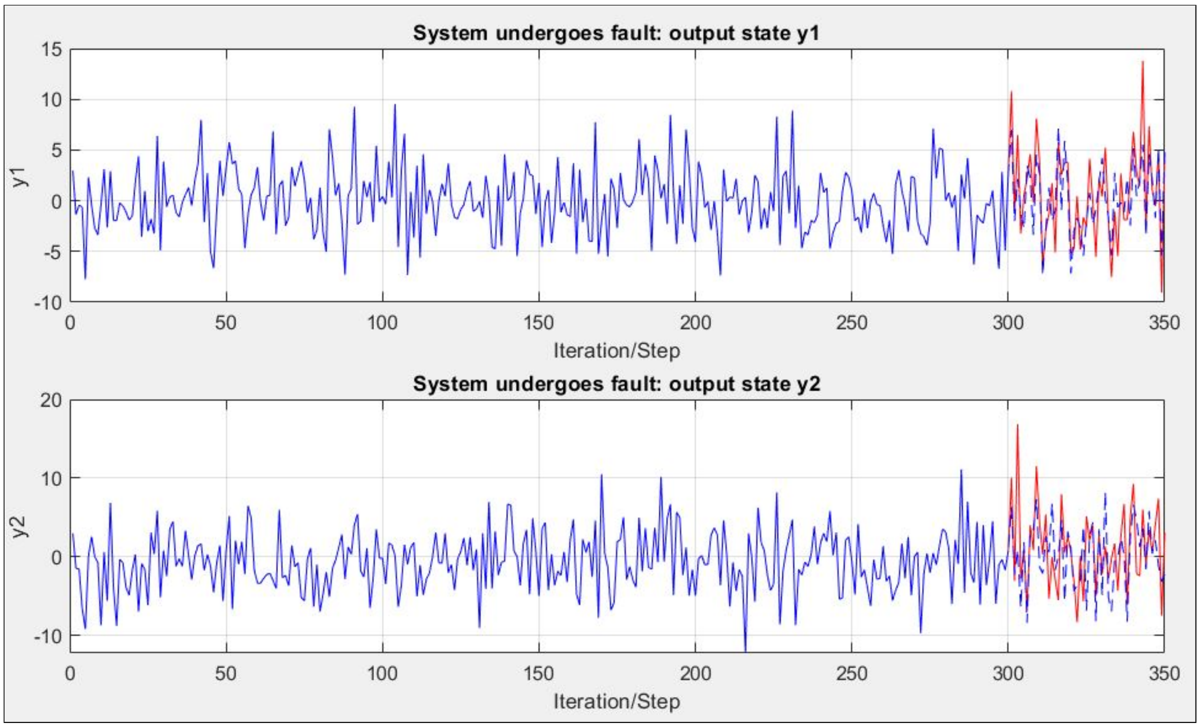


Figure 4.3: Experiment 1: Effect of the fault on the linear system's output states y_1 (top) and y_2 (bottom).

It is expected for the neural network implemented online to carry out the following three tasks:

- Detect the occurrence of the fault in the system from time-step 301 onward (fault detection)

- Isolate the the weights that changed due to the occurrence of the fault (fault isolation)
- Identify the new values of these changed weights (fault identification)

Step 4: Fault diagnosis of faulty system
Figure 4.4 shows the monitoring task performed by the fault-detection algorithm for both output states y_1 (top) and y_2 (bottom) at each time-step. Here, the dotted black lines represent the static upper and lower thresholds, while the solid red line represents the residuals. Figure 4.5 shows the classification made by the fault-detection algorithm at each time-step for both the output states of the system y_1 (left) and y_2 (right). A value of '0' indicates that the system is not at fault while a value of '1' indicates that the system is at fault. From figure 4.5 it can be seen that there are occurrences of a few false positives. However, they are largely spread out and do not occur over successive time-steps. Moreover, for each of them, majority (at least 4 out of 5) of the successive time-steps following the false-positive fail to satisfy the fault detection condition. As a result, the fault detection algorithm treats them as single-point outlier. In contrast, the fault detection algorithm correctly detects the occurrence of a fault in the system's output states y_1 and y_2 from time-step 401 onward, because the majority (at least 4 out of 5) of successive time-steps also satisfy the fault detection condition. The classification accuracy of the fault detection algorithm has been summarized in table 4.2. The classification accuracy for both states was found to be 96%.

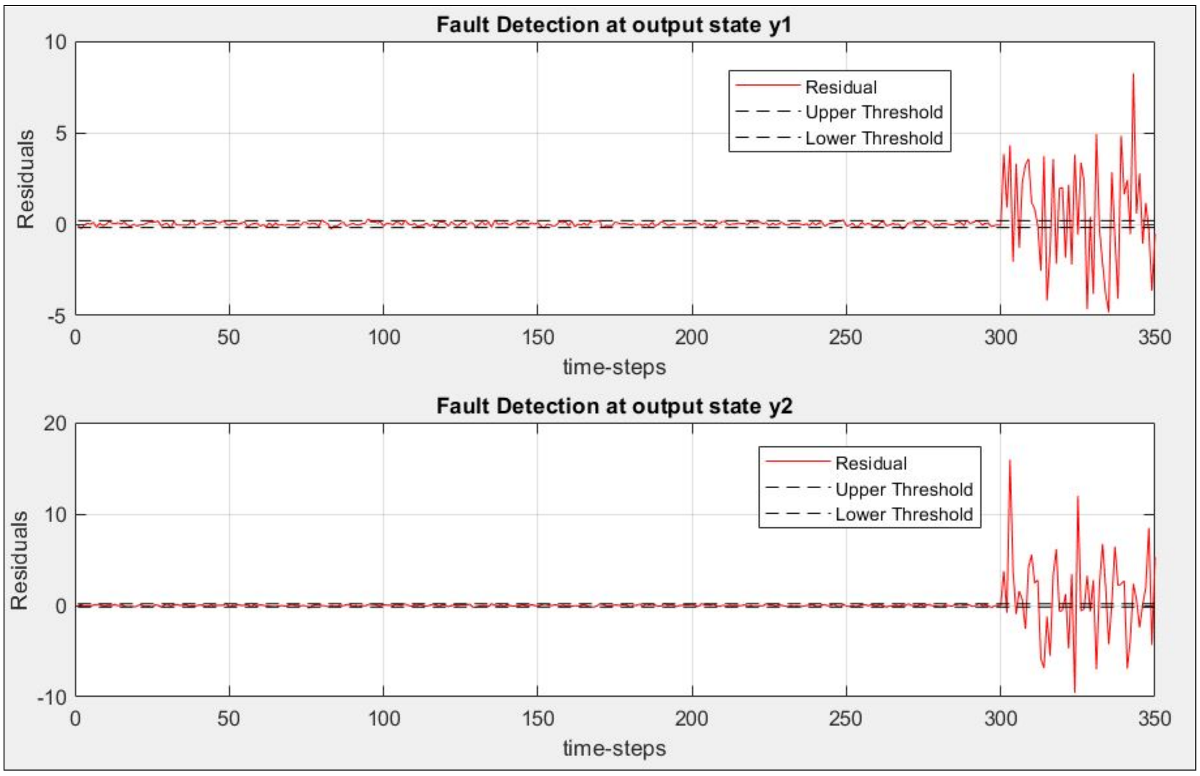


Figure 4.4: Experiment 1: Fault detection algorithm monitoring for faults at the output states y_1 (top) and y_2 (bottom) of the system.

Output State	False positives (out of 350)	False negatives (out of 350)	Correct classifications (out of 350)	Classification accuracy (%)
y1	13	1	336	96
y2	12	2	336	96

Table 4.2: Experiment 1: Classification results of the fault detection algorithm.

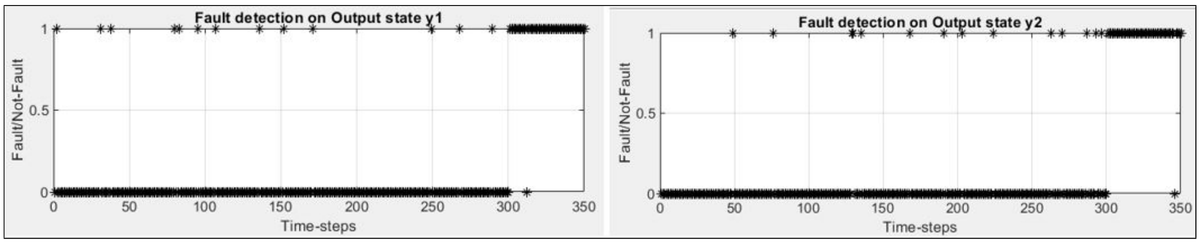


Figure 4.5: Experiment 1: Classification made by the fault detection algorithm for each time-step for output states y_1 (left) and y_2 (right).

Following the detection of the fault, data from the faulty system (301-350) is collected and used to reconstruct the weights by forming an SSR problem between the neurons of the input and output layers and is solved using the sparse Bayesian learning algorithm (algorithm 1). The reconstructed weights are listed in table 4.3 (column 4). From the reconstructed weights, it is clear to see that weights NN_1 and NN_9 (in red) had changed after the occurrence of the fault in the system and are hence isolated. Moreover, close agreement with the true value of the changed-weights (column 3) i.e.low reconstruction error (column 5) shows that the SBL based weight reconstruction approach was successfully able to carry out fault identification of the faulty system.

Weight #	Trained weights	Changed weights	Reconstructed weights	Reconstruction error (%)
NN 1	0.4940	3.4940	3.4949	0.025
NN 2	1.9917	1.9917	1.9810	0.54
NN 3	-1.1074	-1.1074	-1.0935	1.26
NN 4	0.9397	0.9397	0.9321	0.81
NN 5	0.4956	0.4956	0.4952	0.087
NN 6	-2.0013	-2.0013	-1.9887	0.63
NN 7	2.9967	2.9967	3.0022	0.18
NN 8	2.4912	2.4912	2.4798	0.46
NN 9	0.98229	-4.0177	-3.9890	0.71
NN 10	0.49704	0.4970	0.5004	0.68
NN 11	-0.49772	-0.4977	-0.4635	6.87
NN 12	1.4953	1.4953	1.4913	0.27

Table 4.3: Reconstructed weights for experiment 1.

4.2. Experiment 2: Linear system modelled by neural network with a hidden layer

Consider a linear system that that has six input states x_1 to x_6 and two output states y_1 y_2. Let the normal operating dynamics of the linear system be described by equation 4.6, where γ_1 & γ_2 are the measurement noise at output states y_1 and y_2, respectively, and are assumed to be Gaussian.

$$\begin{aligned} y_1 &= 2.5x_1 - 1.5x_2 - 2x_3 + 3x_4 + x_5 + 2x_6 + \gamma_1 \\ y_2 &= -x_1 + 3x_2 + 4x_3 + 2.75x_4 - 1.5x_5 + 3.5x_6 + \gamma_2 \end{aligned} \tag{4.6}$$

Step 1: Data generation from the linear system
The first step is to generate data corresponding to the normal operating conditions of the linear system from its true equation 4.6. Similar to experiment 1, all input states (x_1 to x_6) are Gaussian $\mathcal{N}(0,1)$ with mean = 0 and variance = 1. The measurement noises γ_1 & γ_2 are also assumed to be Gaussian $\mathcal{N}(0,0.01)$ with mean = 0 and variance = 0.01. Figure 4.6 shows the generated training data set for the input (left) and output (right) states of the system.

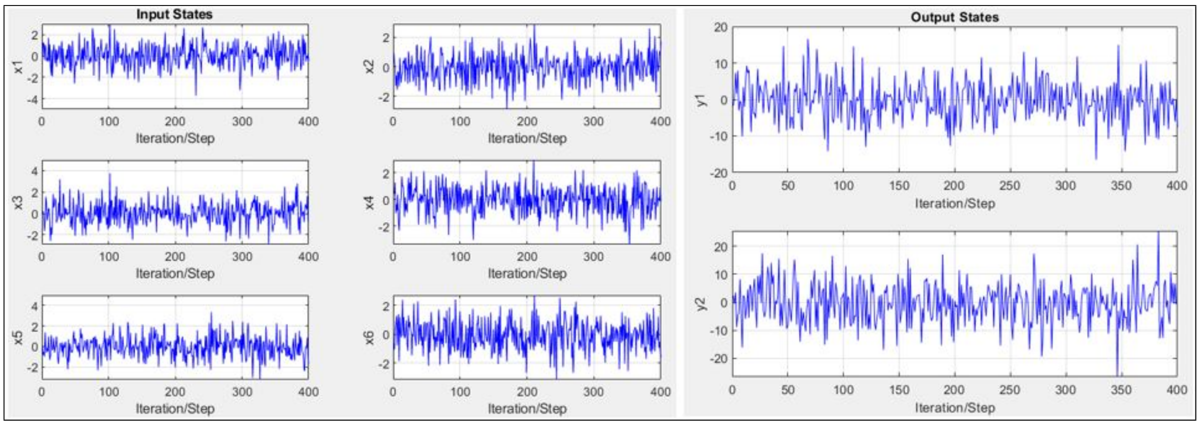


Figure 4.6: Experiment 2: Training data set corresponding to the linear system's six inputs (left) and two output states (right) under normal operating conditions.

Step 2: Modelling linear system with neural network
The neural network architecture chosen to model the linear system is shown in figure 3.5. The neural network's input layer has six neurons, each corresponding to one of the input states of the system (x_1 to x_6). The hidden layer contains two neurons for which the activation function is a linear one. Finally, the output layer has two neurons, each corresponding to one of the output states (y_1 and y_2) of the system. The next step is to train this neural network on the training data set. Once trained, the neural network is cross-validated on the training and test data sets as shown in figure 4.7. Here the red line and the blue stars represent the predicted and target outputs respectively. The mean-square errors (MSE) obtained on the training and test data sets are listed in table 4.4. As expected, the neural network accurately models the linear system. Moreover, it is able to generalize to data it has not been trained on, as is evident from its modelling accuracy on the test data set. The trained weights of this network are given in table 4.7 (third column). Having trained the neural network, the linear system can now be represented as a function of the neural network's weights, given by equation 3.17 where the activation function $f(Z) = Z$ is a linear one.

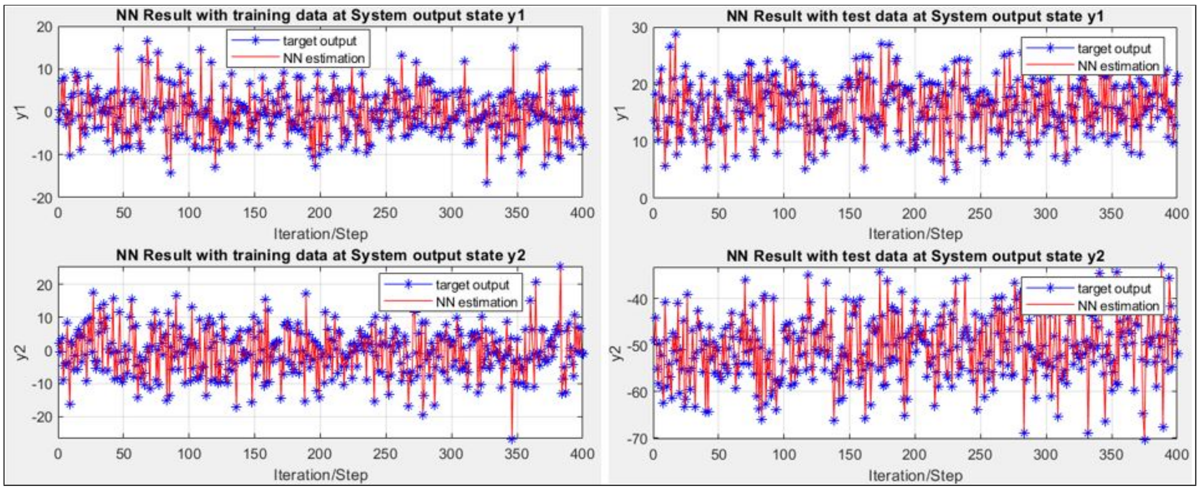


Figure 4.7: Experiment 2: Cross-validation of the trained neural on training (left) and test data sets (right).

Similar to the previous experiment, the low value of the MSE obtained on the training data set indicates that the residual r_i for each data point 'i' in the training data set is either very low or negligible in magnitude. This will result in a low variance of the upper and lower thresholds calculated by the adaptive threshold model and therefore they will behave similar to static thresholds. Hence, for this experiment, a static threshold was implemented instead.

Output State	MSE on training data set	MSE on test data set
y1	0.0111	0.0163
y2	0.0099	0.0349

Table 4.4: Experiment 2: Mean-squared error on the training and test data sets.

The upper and lower thresholds are calculated as per equations 4.2 and 4.3, respectively. Due to the absence of the model (y_e), v is assigned the standard deviation of the residuals. This was found to be 0.1048 and 0.0991 at the system's output states y_1 and y_2, respectively. For a 95% confidence level, t_β was assigned a value of 1.96. The upper and lower thresholds were calculated to be ± 0.2054 and ± 0.1942 for the system's output states y_1 and y_2, respectively.

Step 3: Occurrence of a fault in the system
After the neural network has been trained offline, it is ready for its online implementation alongside the linear system, where it will monitor and detect the occurrence of faults and perform fault identification and isolation tasks after a fault has been detected in the system. At time-step t = 401 during it's operation, a fault in the system occurs as a result of which the weights NN_1 and NN_{13} change their values to $NN_1^{'}$ and $NN_{13}^{'}$, as per equation 4.7. From time-step 401 onward, the faulty system generates data as per equation 4.8. Figure 4.8 shows the effect of the fault on the system's output states y_1 (top) and y_2 (bottom). Here the blue and red plots represents the neural network's predicted output and the system's output (after fault), respectively.

$$\begin{aligned} NN_1^{'} &= NN_1 + 5 \\ NN_{13}^{'} &= NN_{13} + 3 \end{aligned} \tag{4.7}$$

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = f\left(\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{bmatrix} \times \begin{bmatrix} NN_1^{'} & NN_7 \\ NN_2 & NN_8 \\ NN_3 & NN_9 \\ NN_4 & NN_{10} \\ NN_5 & NN_{11} \\ NN_6 & NN_{12} \end{bmatrix} \right) \times \begin{bmatrix} NN_{13}^{'} & NN_{15} \\ NN_{14} & NN_{16} \end{bmatrix} \tag{4.8}$$

It is expected for the neural network implemented online to carry out the following three tasks:

- Detect the occurrence of the fault in the system from time-step 401 onward (fault detection)
- Isolate the the weights that changed (NN_1 and NN_{13}) after the occurrence of the fault (fault isolation)
- Identify the new values of these changed weights (fault identification)

Step 4: Fault diagnosis of faulty system
Figure 4.9 shows the monitoring task performed by the fault-detection algorithm for both output states y_1 (top) and y_2 (bottom) at each time-step. Here, the dotted black lines represent the static upper and lower thresholds, while the solid red line represents the residuals. Figure 4.10 shows the classification made by the fault-detection algorithm at each time-step for both the output states of the system y_1 (left) and y_2 (right). A value of '0' indicates that the system is not at fault while a value of '1' indicates that the system is at fault. From figure 4.10 it can be seen that there are occurrences of a few false positives. However, they are largely spread out

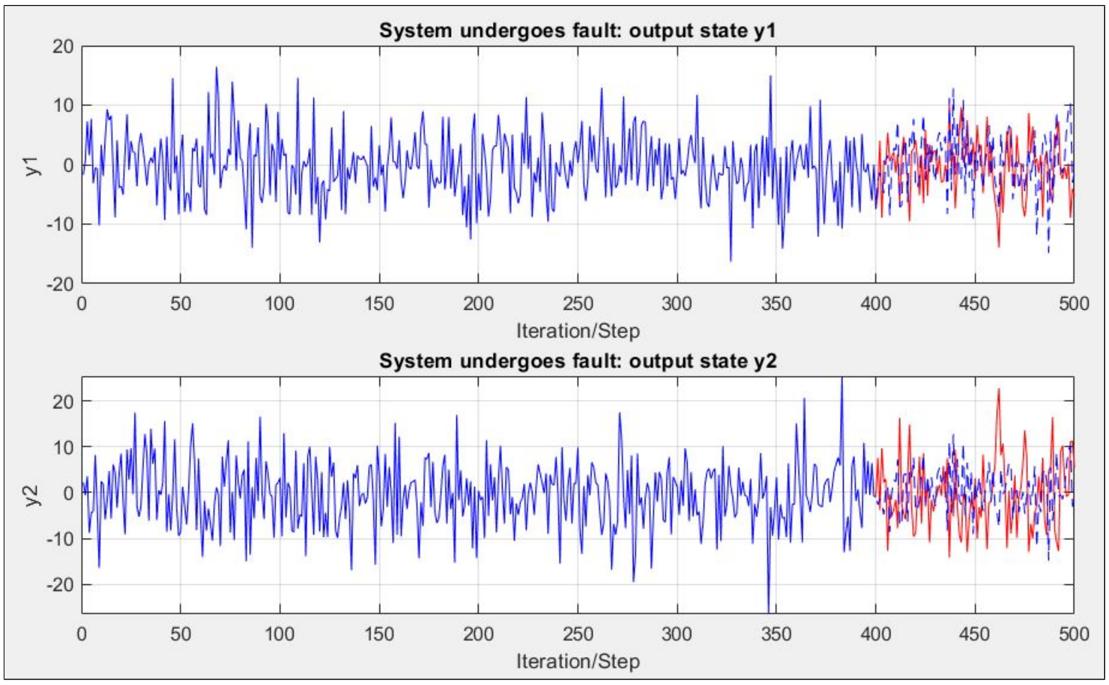


Figure 4.8: Experiment 2: Effect of the fault on the linear system's output states y_1 (top) and y_2 (bottom).

and do not occur over successive time-steps. Moreover, for each of them, majority (at-least 4 out of 5) of the successive time-steps following the false-positive fail to satisfy the fault detection condition. As a result, the fault detection algorithm treats them as single-point outlier. In contrast, the fault detection algorithm correctly detects the occurrence of a fault in the system's output states y_1 and y_2 from time-step 401 onward, because the majority (at-least 4 out of 5) of successive time-steps also satisfy the fault detection condition. The classification accuracy of the fault detection algorithm has been summarized in table 4.5. The classification accuracy at output states y_1 and y_2 was found to be 95% and 94.4%, respectively.

Output State	False positives (out of 500)	False negatives (out of 500)	Correct classifications (out of 500)	Classification accuracy (%)
y1	23	2	475	95
y2	26	2	472	94.4

Table 4.5: Experiment 2: Classification results of the fault detection algorithm.

Following the detection of the fault, the data from the faulty system from time-step 401-500 is used for the reconstruction of the weights. Recall from the previous chapter that for a neural network architecture having a hidden layer, the reconstruction process takes place in stages, i.e., separate reconstruction for the set of weights in W_1 and W_2. This requires estimating the values of the neurons in the hidden layer from time-step 401-500. This requires making an educated guess on the location of the changed-weights by identifying a pattern in the values of the weights of the new neural networks that are trained on the prediction errors at output states y_1 and y_2, respectively. Figure 4.11 shows the prediction errors (in red) on the system's output states y_1 (top) and y_2 (bottom). Figure 4.12 shows the training of the two new neural networks on these prediction errors. The value of the weights of both newly trained neural networks are listed in table 4.6. Looking at the values of the weights in the first neural network, it can be concluded that a changed-weight is located in W_2 of the neural network modelling the linear system and is either one of weights going from the hidden layer neurons to the first output neuron in the output layer i.e. NN_{13} or NN_{14}. Similarly, from

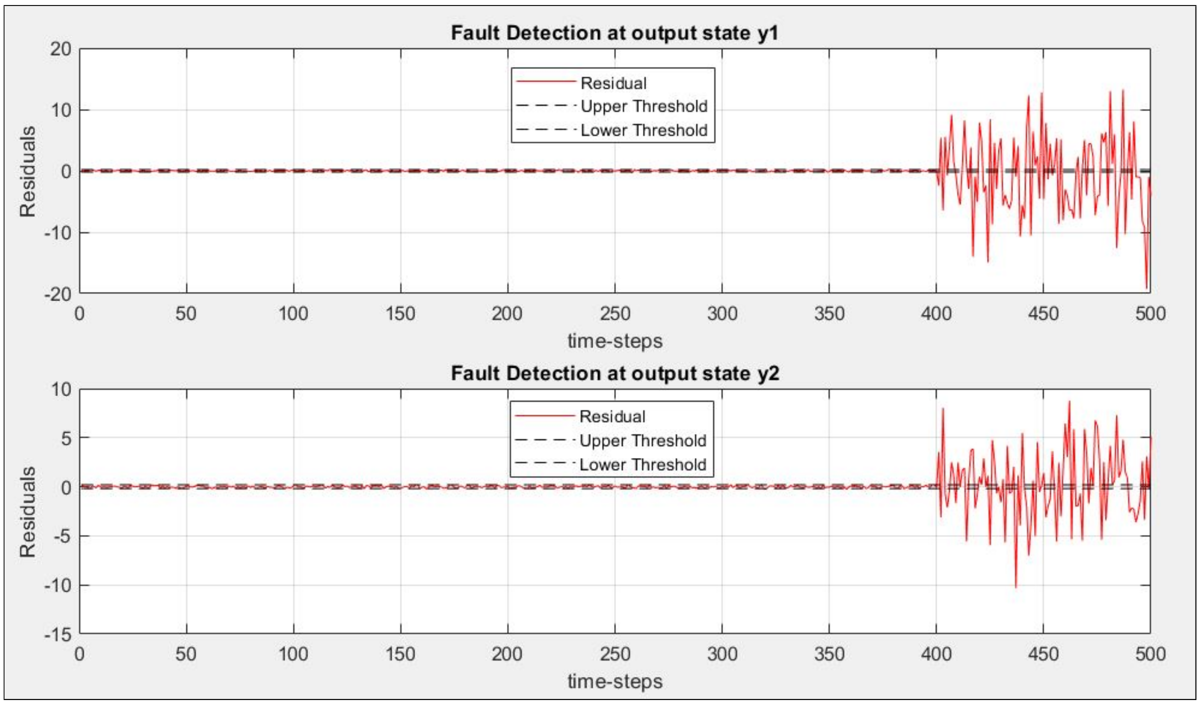


Figure 4.9: Experiment 2: Fault detection algorithm monitoring for faults at the output states y_1 (top) and y_2 (bottom) of the system.

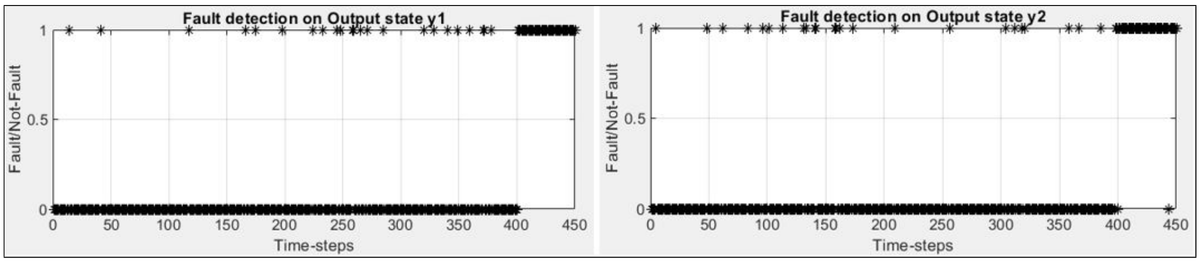


Figure 4.10: Experiment 2: Classification made by the fault detection algorithm for each time-step for output states y_1 (left) and y_2 (right).

the value of the weights in the second neural network, it can be concluded that a second changed-weight is located in W_1 of the neural network modelling the linear system and is either one of weights going from the input layer neurons to the first neuron in the hidden layer i.e. NN_1 or NN_7.

The possible locations of the changed-weights have been identified to be in both W_1 and W_2. Recall similarity to case 2 situation 3 from previous chapter. Hence, in order to reconstruct the changed-weights, the first step is to treat weights NN_1 and NN_7 in W_1 as variables (remaining weights in W_1 remain fixed to their original values) and carry out a single feed-forward step in order to obtain the values of neurons in the hidden layer as a function of these variables. This is followed by the reconstruction of the weights $NN_{13}^{'}$ and $NN_{14}^{'}$ in W_2 by formulating a sparse signal recovery problem between the neurons of the hidden layer and the first neuron of the output layer. This SSR problem is solved using the sparse Bayesian learning algorithm (algorithm 1). Now that the weights $NN_{13}^{'}$ and $NN_{14}^{'}$ in the W_2 layer have been reconstructed, the values of the neurons in the hidden layer for time-steps 401 to 500 are calculated by solving the pairs of linear equations at each time-step. Once estimated, the reconstruction of the neural network's weights $NN_1^{'}$ and $NN_7^{'}$ in W_1 is carried out by formulating another sparse signal recovery problem between the neurons of the input and hidden layers that is solved by applying the sparse Bayesian learning algorithm (algorithm 1). The reconstructed

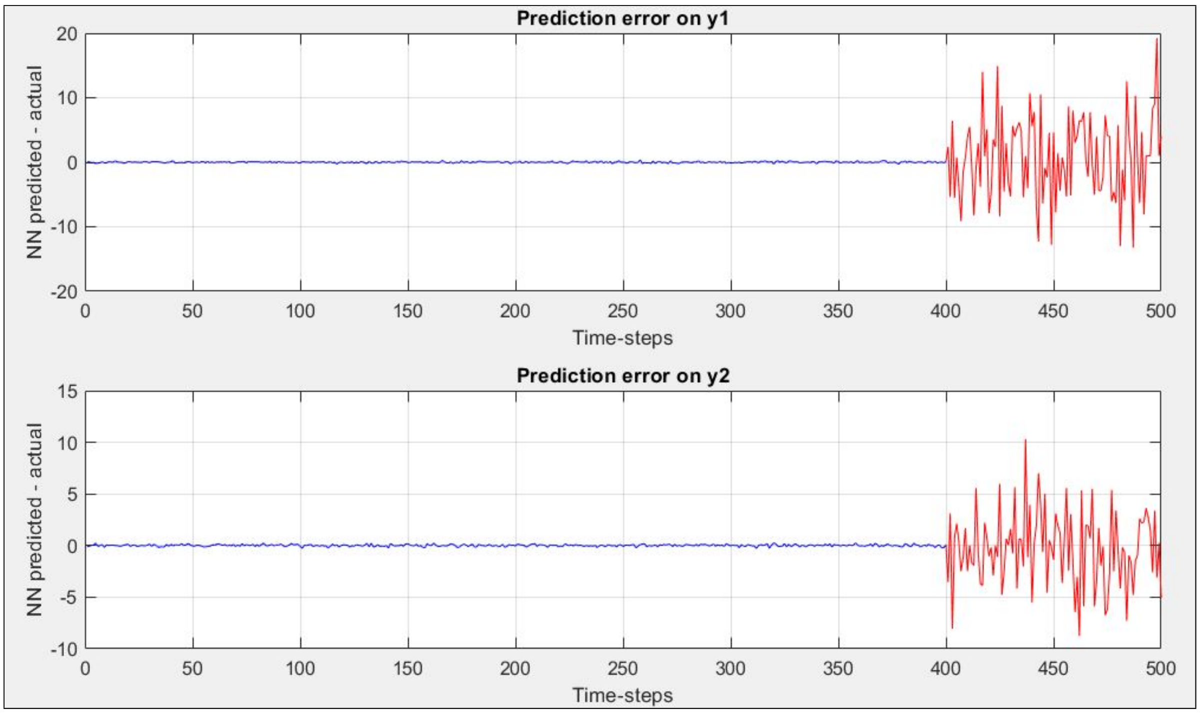


Figure 4.11: Experiment 2: Prediction errors on the output states y_1 (top) and y_2 (bottom) of the system.

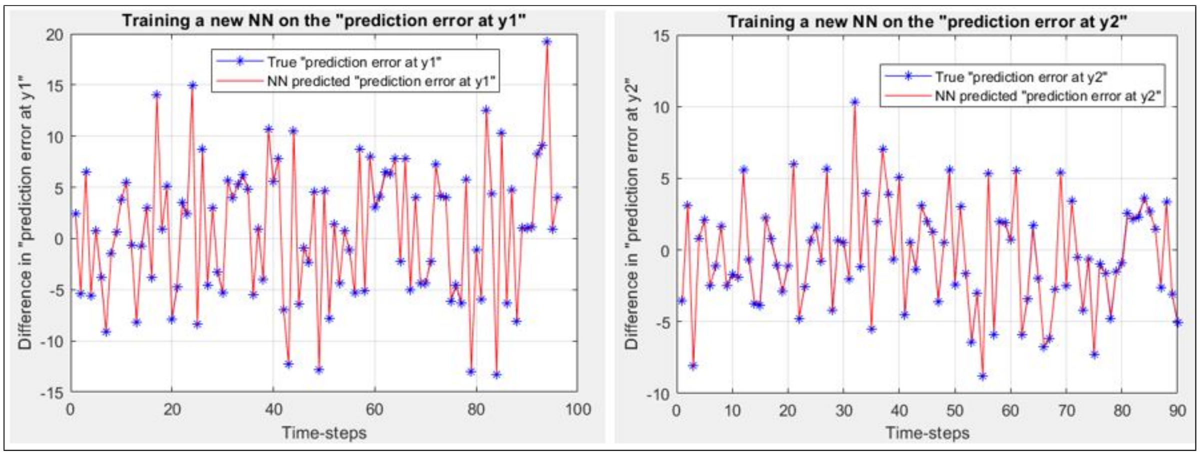


Figure 4.12: Experiment 2: Neural networks trained on the prediction errors on output states y_1 (left) and y_2 (right).

weights are listed in table 4.7 (column 5). From the four reconstructed weights i.e. $NN_{13}^{'}$, $NN_{14}^{'}$, $NN_{1}^{'}$ and $NN_{7}^{'}$ (as all other weights in W_1 and W_2 were kept fixed in value), it is clear to see that the weights NN_1 and NN_{13} (in red) had changed after the occurrence of the fault in the system and are hence isolated. Moreover, their (reconstructed weights) close agreement with the true value of the changed-weights (column 4) i.e. low reconstruction error (column 6) shows that the SBL based weight reconstruction approach was successfully able to carry out the fault identification of the faulty system.

Column 7 demonstrates the importance of having an intuition of the possible locations of the changed-weights by recognizing the pattern in the values of the weights of the neural networks trained on the prediction error. The reconstruction for the weights in this column are done directly i.e. all the weights in W_1 are treated as variables and a single feed-forward step is carried out in order to obtain the values of neurons in the hidden layer as a function of these variables. The weights in W_2 are then reconstructed by formulating two SSR prob-

Weight #	Trained weights (neural network 1)	Trained weights (neural network 2)
NN 1	-1.3583	3.7128
NN 2	-0.5477	0.0078
NN 3	-0.7544	-0.0315
NN 4	4.8121	0.0063
NN 5	0.5211	-0.0024
NN 6	3.8699	-0.0191

Table 4.6: Experiment 2: Value of the weights of the two new neural networks trained on the prediction errors on the output states of the system.

Weight lies in (W1/W2)	Weight #	Trained weights	Changed weights	Reconstructed weights (with intuition)	Reconstruction error (%)	Reconstructed weights (without intuition)
W1	NN 1	-0.8751	4.129	4.1361	0.27	-1.7194
	NN 2	0.1771	0.1771	0.1771	N/A	-2.3323
	NN 3	0.2411	0.2411	0.2411	N/A	-3.1681
	NN 4	-1.6110	-1.6110	-1.6110	N/A	-0.4934
	NN 5	-0.1861	-0.1861	-0.1861	N/A	1.2736
	NN 6	-1.3054	-1.3054	-1.3054	N/A	-1.3248
	NN 7	-0.6612	-0.6612	-0.65166	1.44	-1.8918
	NN 8	1.2596	1.2596	1.2596	N/A	0.0254
	NN 9	1.6797	1.6797	1.6797	N/A	0.0200
	NN 10	0.6170	0.6170	0.6170	N/A	0.7687
	NN 11	-0.6629	-0.6629	-0.6629	N/A	0.0221
	NN 12	1.0112	1.0112	1.0112	N/A	0.6559
W2	NN 13	-2.1966	0.8034	0.8013	0.26	0.3870
	NN 14	-0.8725	-0.8725	-0.87722	0.54	-2.1366
	NN 15	-0.7500	-0.7500	-0.7500	N/A	-1.2450
	NN 16	2.4820	2.4820	2.4820	N/A	2.7657

Table 4.7: Reconstructed weights for experiment 2.

lems between the neurons of the hidden layer and the first neuron of the output layer. The first one for reconstructing weights NN'_{13} and NN'_{14}, while the second one for reconstructing weights NN'_{15} and NN'_{16}. The reconstructed weights are used to calculate the values of the neurons in the hidden layer. Once estimated, the weights in W_1 are reconstructed by formulating another 2 SSR problems between the neurons of the input and hidden layers. The first one for reconstructing weights NN'_1 to NN'_6, while the second one is for reconstructing weights NN'_7 to NN'_{12}. It can be seen that set of reconstructed weights in W_1 and W_2 are both highly inaccurate to the true values of the weights after the fault (column 4).

Note: *The optimization problem in the SBL algorithm is solved in MATLAB using the fminsearch function. fminsearch requires a starting or initial value for the variables that are being optimized. In this context, the variables being optimized are the set of weights in the neural network which are suspected (intuition from pattern) to have changed after the occurrence of the fault in the system i.e. W_1 and W_7 in this experiment. Hence, while solving the optimization problem, an initial starting value for these to-be reconstructed weights are to be provided. The provide initial values for these weights were their originally trained values from table 4.7 (column 3).*

4.3. Experiment 3: Nonlinear system modelled by neural network with a hidden layer

Consider a nonlinear system that that has six input states x_1 to x_6 and two output states y_1 and y_2. Let the normal operating dynamics of the nonlinear system be described by equation 4.9, where γ_1 and γ_2 are the measurement noise at output states y_1 and y_2, respectively that are assumed to be Gaussian.

$$\begin{aligned} y_1 &= x_1^2 + 2x_2 - 3x_3 + x_4 - x_5 - 2.5x_6 + \gamma_1 \\ y_2 &= x_1 + 3.5x_2 - x_3^2 + 1.5x_4 - 0.5x_5 - x_6 + \gamma_2 \end{aligned} \tag{4.9}$$

Step 1: Data generation from the nonlinear system
The first step is to generate data corresponding to the normal operating conditions of the nonlinear system from its true equation 4.9. Similar to the previous experiment, all input states (x_1 to x_6) are Gaussian $\mathcal{N}(0, 1)$ of mean = 0 and variance = 1. The measurement noises γ_1 and γ_2 are also assumed to be Gaussian $\mathcal{N}(0, 0.01)$ of mean = 0 and variance = 0.01. Figure 4.13 shows the generated training data set for the input (left) and output (right) states of the system.

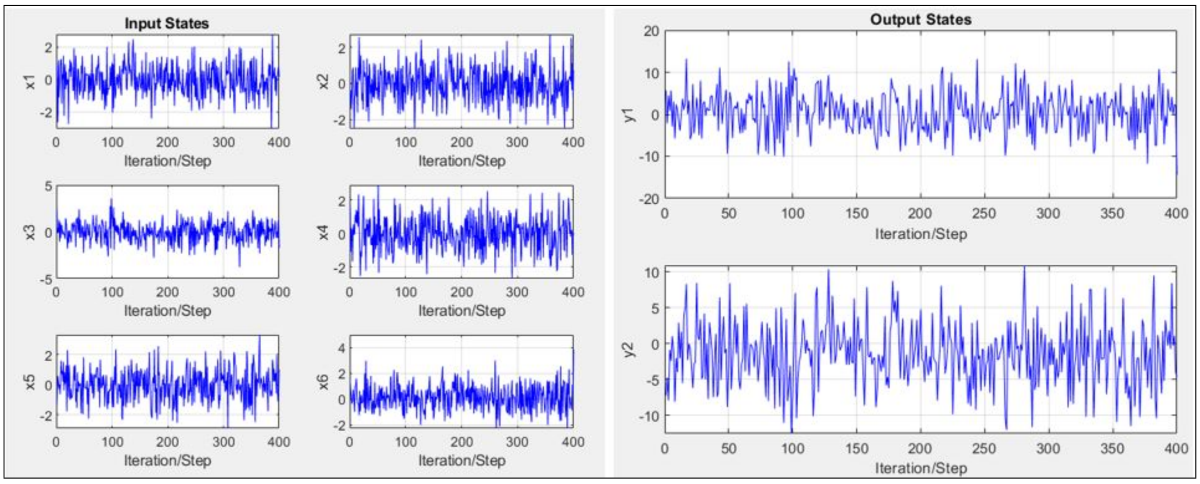


Figure 4.13: Experiment 3: Training data set corresponding to the linear system's six inputs (left) and two output states (right) under normal operating conditions.

Step 2: Modelling nonlinear system with neural network
The neural network architecture chosen to model the nonlinear system is shown in figure 3.5. The neural network's input layer has six neurons, each corresponding to one of the input states of the system (x_1 to x_6). The hidden layer contains two neurons for which the activation function is chosen to be the sigmoid function (nonlinear). Finally, the output layer has two neurons, each corresponding to one of the output states (y_1 and y_2) of the system. The next step is to train this neural network on the training data set. Once trained, the neural network is cross-validated on the training and test data sets as shown in figure 4.14. Here the red line and the blue stars represent the predicted and target outputs respectively.The MSE obtained on the training and test data sets are listed in table 4.8. The high MSE value on the training data set indicates that the neural network is unable to accurately model the nonlinear system. Moreover, its does not show a good ability to generalize on data it has not trained on as is indicated by the MSE on the test data set. Recall that in order to obtain a deterministic set of equations for calculating the values of neurons in the hidden layer, the number of neurons in the hidden and outer layers must be equal. As a result of this constraint, the choice for modelling the system is limited to the current neural network architecture or perhaps an architecture where there a several hidden layers, each having two neurons. However, with multiple hidden layers, identifying the pattern becomes more challenging. Hence, the

current architecture is the preferred choice of modelling the nonlinear system in this case. The trained weights of this network are given in table 4.11 (third column). Having trained the neural network, a reduced version of the nonlinear system can be represented as a function of the neural network's weights, given by equation 3.17 where the activation function ($f(Z)$) is the sigmoid function.

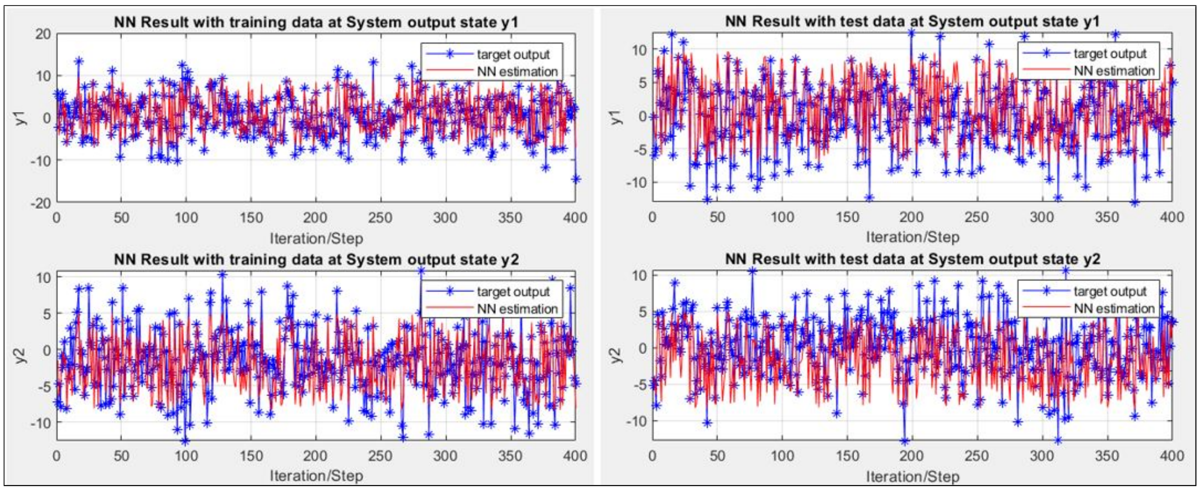


Figure 4.14: Experiment 3: Cross-validation of the trained neural on the training (left) and test (right) data sets.

Output State	MSE on training data set	MSE on training data set
y1	5.4314	7.8919
y2	8.1848	11.3033

Table 4.8: Experiment 3: Mean-squared error on the training and test data sets.

Unlike the previous experiments, in this experiment a high value of MSE was obtained on the training data set as a result of which the residuals r_i for each data point 'i' in the training data set are quite significant in their magnitude. Hence, neural networks are trained on the input-residual pair data sets $((u_i, r_i)_{i=1}^{N})$ in order to construct adaptive threshold models for each output state of the system. Figure 4.15 shows the trained adaptive threshold models for output states y_1 (left) and y_2 (right), respectively.

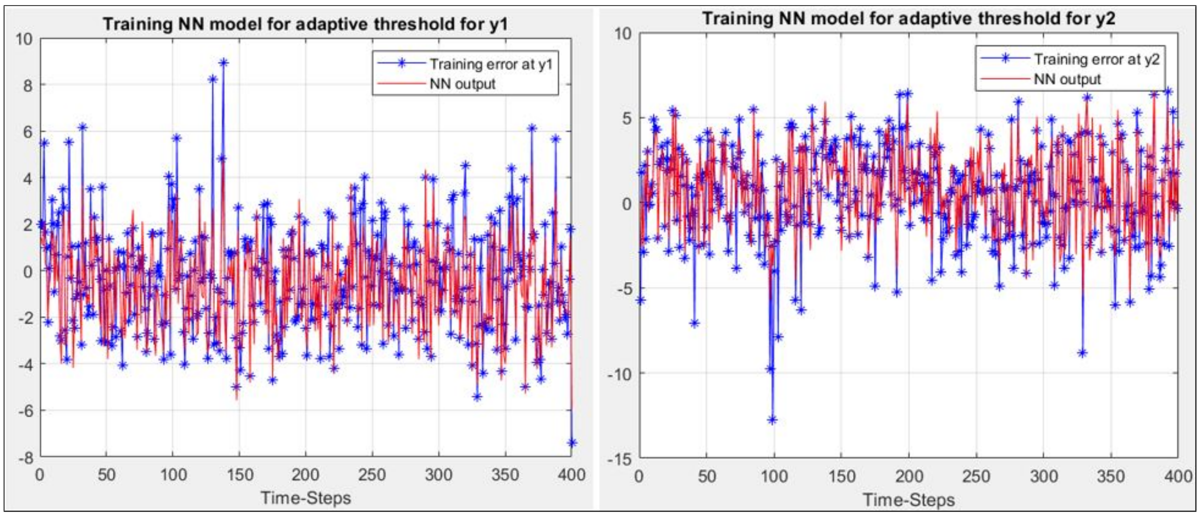


Figure 4.15: Experiment 3: Trained adaptive threshold models for output states y_1 (left) and y_2 (right).

Step 3: Occurrence of a fault in the system
After the neural network has been trained offline, it is ready for its online implementation alongside the nonlinear system, where it will monitor and detect the occurrence of faults and perform fault identification and isolation tasks after a fault has been detected in the system. At time-step t = 401 during it's operation, a fault in the system occurs as a result of which the weights NN_1 and NN_{13} change their values to NN_1' and NN_{13}', as per equation 4.10. From time-step 401 onward, the faulty system generates data as per equation 4.11. Figure 4.16 shows the effect of the fault on the system's output states y_1 (top) and y_2 (bottom). Here the blue and red plots represents the neural network's predicted output and the system's output (after fault), respectively.

$$\begin{aligned} NN_1' &= NN_1 + 4 \\ NN_{13}' &= NN_{13} + 20 \end{aligned} \tag{4.10}$$

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = f\left(\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{bmatrix} \times \begin{bmatrix} NN_1' & NN_7 \\ NN_2 & NN_8 \\ NN_3 & NN_9 \\ NN_4 & NN_{10} \\ NN_5 & NN_{11} \\ NN_6 & NN_{12} \end{bmatrix} \right) \times \begin{bmatrix} NN_{13}' & NN_{15} \\ NN_{14} & NN_{16} \end{bmatrix} \tag{4.11}$$

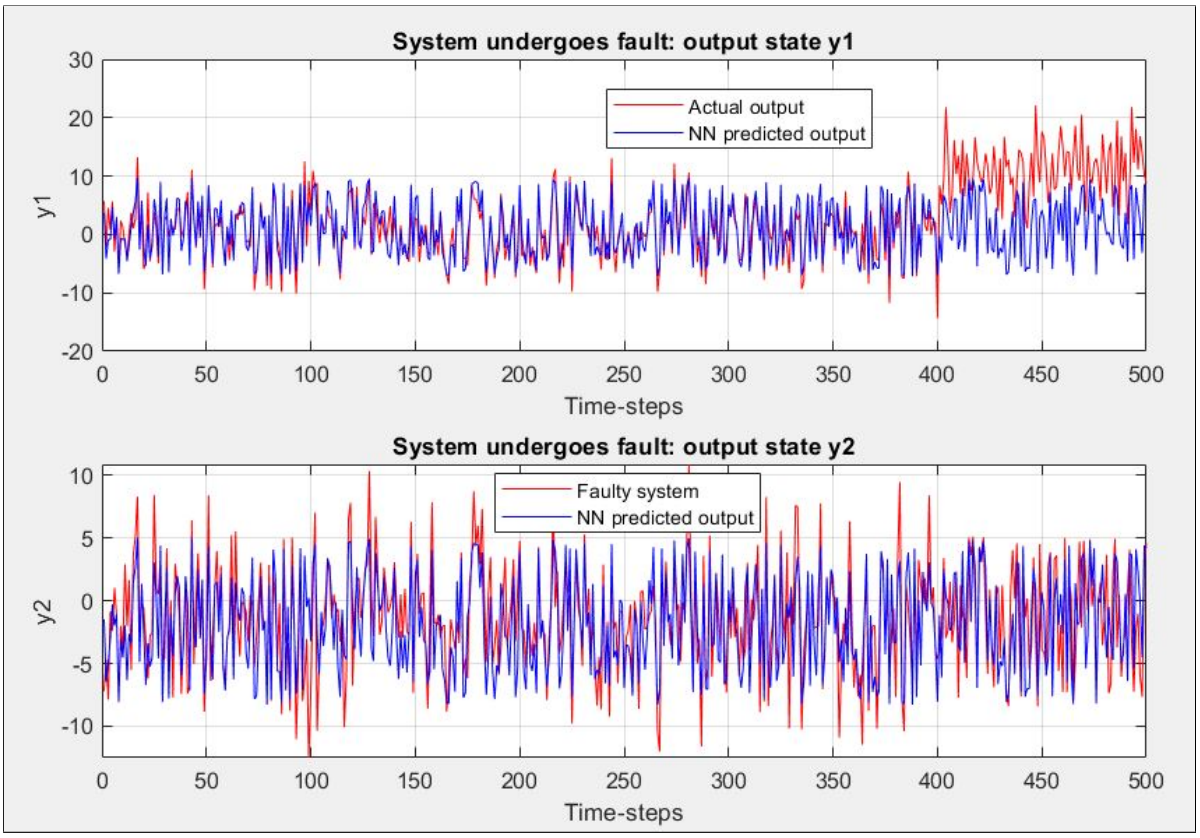


Figure 4.16: Experiment 3: Effect of the fault on the nonlinear system's output states y_1 (top) and y_2 (bottom).

It is expected for the neural network implemented online to carry out the following three tasks:

- Detect the occurrence of the fault in the system from time-step 401 onward (fault detection)

- Isolate the the weights that changed (NN_1 and NN_{13}) after the occurrence of the fault (fault isolation)
- Identify the new values of these changed weights (fault identification)

Step 4: Fault diagnosis of faulty system
Figure 4.17 shows the monitoring task performed by the fault-detection algorithm for both output states y_1 (left) and y_2 (right) at each time-step. Here, the dotted black lines represent the static upper and lower thresholds, while the solid red line represents the residuals. Figure 4.18 shows the classification made by the fault-detection algorithm at each time-step for both the output states of the system y_1 (top) and y_2 (bottom). A value of '0' indicates that the system is not at fault while a value of '1' indicates that the system is at fault. From figure 4.18 it can be seen that there are occurrences of more false positives and negatives than the previous experiments. However, they are quite spread out and do not occur over multiple successive time-steps (example 4 or 5 successive time-steps). Moreover, for each of them, majority (at least 4 out of 5) of the successive time-steps following the false-positive fail to satisfy the fault detection condition. As a result, the fault detection algorithm treats them as single-point outlier. In contrast, the fault detection algorithm correctly detects the occurrence of a fault in the system's output state y_1 from time-step 401 onward, because the majority (at-least 4 out of 5) of successive time-steps also satisfy the fault detection condition. The classification accuracy of the fault detection algorithm has been summarized in table 4.9. The classification accuracy at output states y_1 and y_2 were found to be 83.6% and 95%, respectively.

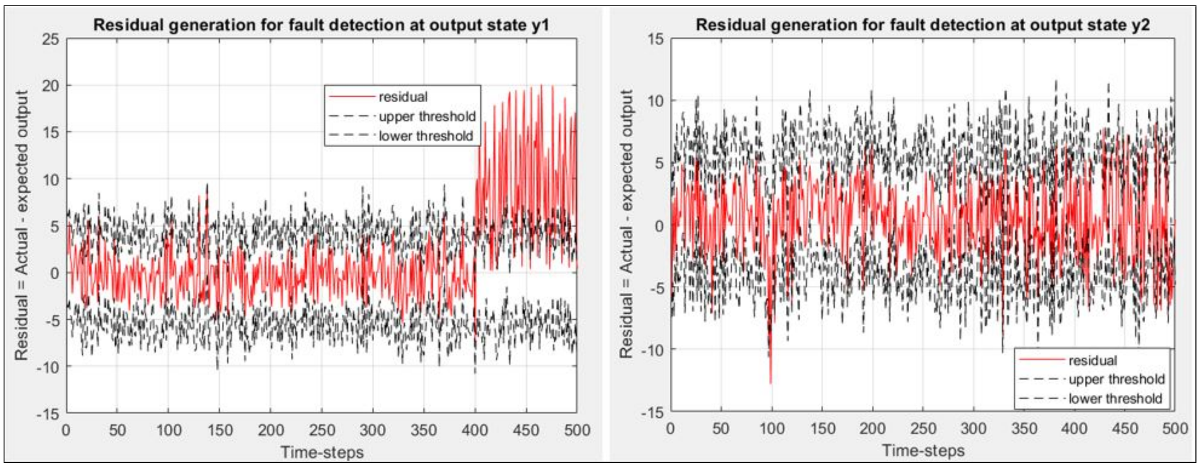


Figure 4.17: Experiment 3: Fault detection algorithm monitoring for faults at the output states y_1 (left) and y_2 (right) of the system.

Output State	False positives (out of 500)	False negatives (out of 500)	Correct classifications (out of 500)	Classification accuracy (%)
y1	61	21	418	83.6
y2	25	0	475	95

Table 4.9: Experiment 3: Classification results of the fault detection algorithm.

Following the detection of the fault, the data from the faulty system from time-step 401-500 is used for the reconstruction of the weights. Before reconstruction, an educated guess on the location of the changed-weights must be made by identifying a pattern in the values of the weights of the new neural networks that are trained on the prediction errors at output states y_1 and y_2. Figure 4.19 shows the prediction errors (in red) on the system's output states y_1 (top) and y_2 (bottom). The value of the weights of both newly trained neural networks are listed in table 4.10.

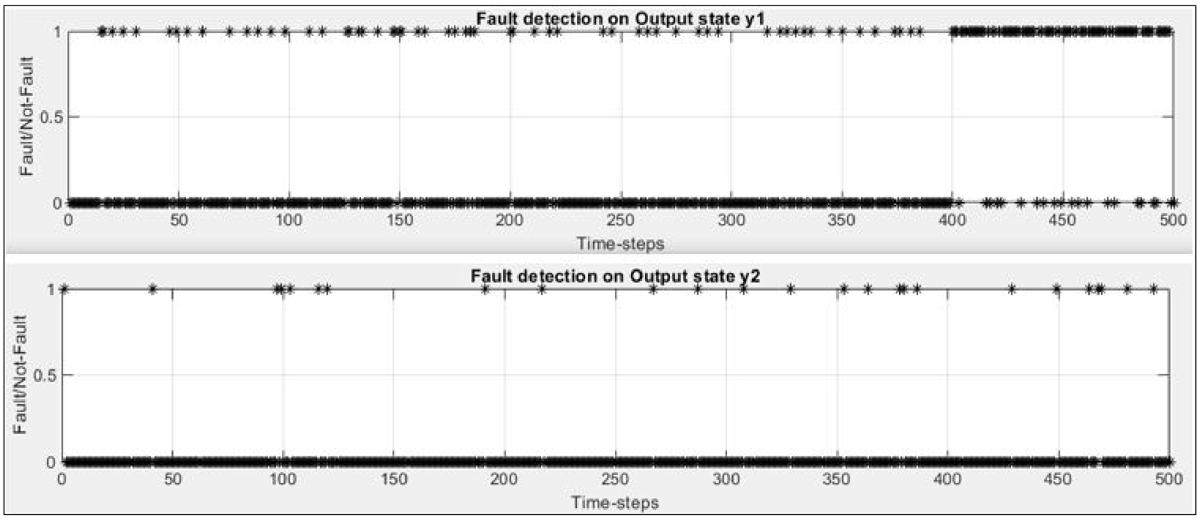


Figure 4.18: Experiment 3: Classification made by the fault detection algorithm for each time-step for output states y_1 (top) and y_2 (bottom).

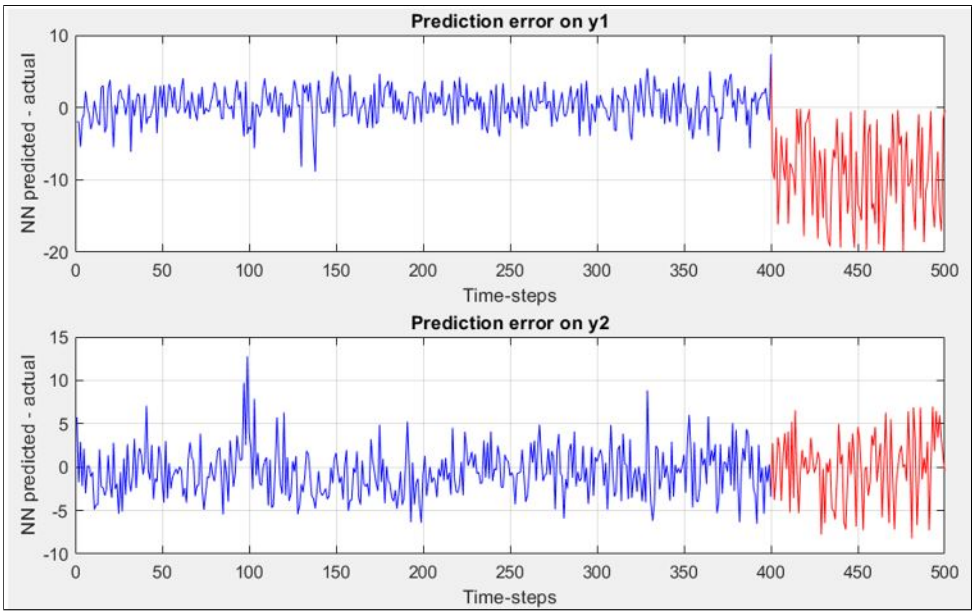


Figure 4.19: Experiment 3: Prediction error on the output states y_1 and y_2 of the system. Part in red is the prediction error after the occurrence of the fault from time-step 401 onward

In the first neural network, for the weights in W_1, with the exception of the first pair of weights (NN_1 & NN_7) the other weight-pairs (NN_2 & NN_8, NN_3 & NN_9, NN_4 & NN_{10}, NN_5 & NN_{11} and NN_6 & NN_{12}) are nearly equal in magnitude and same in sign. Meanwhile the two weights in W_2 vary significantly from each other in magnitude and are same in sign. In the second neural network, for the weights in W_1, all the pairs of weights are significantly different from each other in magnitude. From the combination of these patterns, it can be inferred that a changed-weight is located in W_2 of the neural network modelling the system. The location of this changed weight can be narrowed to one of the weights going from the neurons in the hidden layer to the first neuron in the output layer i.e., NN_{13} or NN_{14}. Moreover, the mismatch in magnitudes of the first pair of weights in the first neural network implies that a second changed-weight is located in W_1. The location of this changed weight can be narrowed to one of the weights going from the first neuron of the input layer to all neurons in the hidden layer i.e., NN_1 or NN_7.

Weight lies in (W1/W2)	Trained weights (neural network 1)		Trained weights (neural network 2)	
W1	3.6021	-0.3782	-2.8016	-1.7093
	-1.6172	-1.6327	-1.2265	0.6859
	0.0072	0.0063	-0.6813	-2.5653
	-0.8125	-0.8086	-2.1275	-0.7716
	0.3516	0.3471	1.7142	-1.4905
	1.0559	1.0889	0.0644	0.1387
W2	-12.6709		-5.8644	
	-7.2947		-1.2304	

Table 4.10: Experiment 3: Identified pattern in the value weights of the two new neural networks that are trained on the prediction errors on the output states (y_1 and y_2) of the system.

The possible locations of the changed-weights have been identified to be in both W_1 and W_2. Recall similarity to case 3 situation 2 from previous chapter. Hence, in order to reconstruct the changed-weights, the first step is to treat weights NN_1 and NN_7 in W_1 as variables (remaining weights in W_1 remain fixed to their original values) and carry out a single feed-forward step in order to obtain the values of neurons in the hidden layer as a function of these variables. This is followed by the reconstruction of the weights NN'_{13} and NN'_{14} in W_2 by formulating a sparse signal recovery problem between the neurons of the hidden layer and the first neuron of the output layer. This SSR problem is solved using the sparse Bayesian learning algorithm (algorithm 1). Now that the weights NN'_{13} and NN'_{14} in the W_2 layer have been reconstructed, the values of the neurons in the hidden layer for time-steps 401 to 500 are calculated by solving the pairs of linear equations at each time-step. Once estimated, the reconstruction of the neural network's weights NN'_1 and NN'_7 in W_1 is carried out by formulating another sparse signal recovery problem between the neurons of the input and hidden layers that is solved by applying the sparse Bayesian learning algorithm for the nonlinear case as shown in algorithm 3. The reconstructed weights are listed in table 4.11 (column 5). From the four reconstructed weights i.e. NN'_{13}, NN'_{14}, NN'_1 and NN'_7 (as all other weights in W_1 and W_2 were kept fixed in value), it is clear to see that the weights NN_1 and NN_{13} (in red) had changed after the occurrence of the fault in the system and are hence isolated. Moreover, their (reconstructed weights) close agreement with the true value of the changed-weights (column 4) i.e. low reconstruction error (column 6) shows that the SBL based weight reconstruction approach was successfully able to carry out the fault identification of the faulty system.

Column 7 shows the results obtained when all weights in W_1 and W_2 are reconstructed directly without building an intuition on the possible location(s) of the changed weights in the neural network modelling the nonlinear system. The process for reconstructing the weights in W_1 and W_2 remains same to the one described in the previous experiment with the single exception that the nonlinear SBL algorithm (algorithm 3) is used for the reconstruction of the weights in W_1 to account for the nonlinear activation function on the hidden layer of the neural network. From column 7 it can be seen that set of reconstructed weights in W_1 and W_2 are both highly inaccurate to the true values of the weights after the fault (column 4).

Note: *The optimization problem in the SBL algorithm is solved in MATLAB using the fminsearch function. fminsearch requires a starting or initial value for the variables that are being optimized. In this context, the variables being optimized are the set of weights in the neural*

Weight lies in (W1/W2)	Weight #	Trained weights	Changed weights	Reconstructed weights (with intuition)	Reconstruction error (%)	Reconstructed weights (without intuition)
W1	NN 1	-0.4008	3.5992	3.4668	3.68	-1.7194
	NN 2	-1.6260	-1.6260	-1.6260	N/A	-1.6014
	NN 3	-0.0053	-0.0053	-0.0053	N/A	-0.2985
	NN 4	-0.8070	-0.8070	-0.8070	N/A	-1.7947
	NN 5	0.3488	0.3488	0.3488	N/A	1.1858
	NN 6	1.0755	1.0755	1.0755	N/A	1.8948
	NN 7	-0.5432	-0.5432	-0.5086	6.37	-0.7300
	NN 8	0.6458	0.6458	0.6458	N/A	-3.8572
	NN 9	1.3559	1.3559	1.3559	N/A	-0.1794
	NN 10	0.5304	0.5304	0.5304	N/A	-1.5624
	NN 11	-0.7602	-0.7602	-0.7602	N/A	0.7266
	NN 12	-1.1962	-1.1962	-1.1962	N/A	1.0057
W2	NN 13	-7.2495	12.7505	12.7462	0.03	-0.2511
	NN 14	9.8016	9.8016	9.8047	0.03	1.3115
	NN 15	-8.3811	-8.3811	-8.3811	N/A	-0.8103
	NN 16	5.0921	5.0921	5.0921	N/A	0.1712

Table 4.11: Reconstructed weights for experiment 3.

network which are suspected (intuition from pattern) to have changed after the occurrence of the fault in the system i.e. W_1 and W_7 in this experiment. Hence, while solving the optimization problem, an initial starting value for these to-be reconstructed weights are to be provided. The provide initial values for these weights were their originally trained values from table 4.11 (column 3).

4.4. Discussion

The goal of this chapter was to investigate the suitability of the proposed fault detection and fault identification-isolation (SBL) algorithms for carrying out a fault diagnosis task. The suitability was tested by undertaking three experiments each differing from one another in the nature of the system (linear or nonlinear) and/or in the neural network architecture with which they have been modelled (with or without hidden layers and with linear or nonlinear activation functions). It was found that for both the linear systems in experiments 1 and 2, the fault diagnosis algorithms were successful in not only detecting the time-step(s) at which the faults occurred (fault detection), but also in accurately reconstructing the new values of the weights in the neural network (fault identification) using the SBL algorithms and correctly locating and isolating the changed-weights (fault isolation). In experiment 2 albeit a linear system, it was shown that even with the presence of a hidden layer, the locations of the changed-weights were identifiable by forming an intuition based on the value of the weights of a neural network trained on the prediction error of the output states y_1 and y_2. This intuition then lead to the estimation of values of the neurons of the hidden layer which were key to the accurate reconstruction of weights using the SBL algorithm. Finally, in experiment 3, the fault diagnosis algorithms were tested on a nonlinear system. The fault detection algorithm performed reasonably well by successfully detecting the occurrence of the fault in the system while managing to return only a few sparse false-positives and false-negatives. The weight reconstruction (fault identification) using the nonlinear variant of the SBL algorithm (algorithm 3) and the fault isolation of the changed-weights were shown to be successful following the forming of an accurate intuition on all the possible locations of the changed-weights. This was based on the observed pattern in the value of weights across W_1 and W_2 layers of the neural networks that were newly trained on the prediction errors of the output states of the system.

5 Conclusions and Future Work

5.1. Conclusions

The goal of this study was to explore the possibility of undertaking a fault diagnosis task for systems modelled using neural networks. This was done by using an adaptive threshold based fault detection algorithm, and sparse Bayesian learning based fault isolation & identification algorithm. Experiments were conducted on (simplified) linear and nonlinear systems using the proposed algorithms, and based on the results obtained the following key conclusions are made:

Fault Detection Algorithm:

- The success of the fault detection algorithm for a system is dependent on the accuracy with which the neural network models the system. For experiments 1 & 2 (linear systems), the trained neural networks were able to model the non-faulty behaviour of the system with high accuracy, which translated to a high accuracy of fault detection (95%). Whereas, for experiment 3 (nonlinear system), the modelling accuracy of the trained neural network was relatively lower, which resulted in more false positives and negatives being detected, and a classification accuracy of 83%.
- It was also noted that for neural networks with high modelling accuracy, a static threshold may be sufficient for fault detection, since the standard deviation of prediction error is negligible. However, for those cases with lower modelling accuracy, adaptive threshold is more suitable since it is able to capture the prediction errors due to the neural network.

Fault Isolation & **Identification Algorithm:**

- The application of fault isolation & identification for linear systems without a hidden layer (experiment 1) was found to be straightforward because of a direct mapping between the neurons of the input and output layers. Due to that, the proposed algorithm using Sparse Bayesian learning was able to reconstruct the faulty weights accurately. However, repeating the same for experiments 2 & 3 resulted in a poor reconstruction due to the presence of a hidden layer. For these systems, it was proposed that an intuition on the possible location(s) of the changed weight(s) could be formed by training a new set of neural networks on the prediction errors of the system's output state(s). Doing so resulted in an observation of distinctive patterns in the values of the weights of these neural networks. Moreover, these patterns were found to give an indication of the true location(s) of the changed-weight(s) in the neural network modelling the system. Using this intuition, for systems with a hidden layer, a stage-wise reconstruction of the neural network weights was undertaken and found to be successful for both linear and non-linear systems with a hidden layer.

This study was a first step towards undertaking fault diagnosis for mechanical systems modelled using neural networks. Overall, the proposed algorithms showed promising results in carrying out fault diagnosis for all the three experiments conducted. However, firstly, the experiments consisted of simplified (hypothetical) mechanical systems. Secondly, the task of relating the weights with the physical properties of an actual mechanical system still remains to be explored. Hence, in order to establish the suitability of applications of the proposed algorithms for real world cases, further research is needed. Some ideas on potential future research directions are detailed in the next section.

5.2. Recommendations for Future Work

Based on the results of this study, the following potential directions for future research are proposed:

1. **Relating the properties of the system to the weights of the modelled neural network.**
   Recall having mentioned earlier that traditionally the weights of a neural network are real numbers of certain magnitude that provide a simple understanding of how positively or negatively sensitive two neurons are to one another. Based on the values of the weights, very limited and basic insights on the system can be gained i.e., a large positive weight implies that the input neuron has a strong proportional relation to the output neuron while a negative weight implies an inverse relation. In the current work, a theoretical neural network framework was considered where the properties of the system being modelled were assumed to be represented by the weights of the neural network. For this theoretical framework, it was shown that with the help of the proposed fault diagnosis algorithms, a complete fault diagnosis of the faulty system is possible.
   However, the current work does not focus on how such a framework can be realized for a given physical system. Hence, a recommended direction for future work is to investigate how such a framework can be realized for any given physical system and how the weights of this neural network can be interpreted in order to gain better insights on the system being modelled.

2. **Extending the fault detection algorithm to detect the future occurrence of faults in the system during its operation.**
   In the current work, the fault diagnosis process is carried out only after the fault has already occurred in the system. However in process & manufacturing engineering domains where the systems are required to operate reliably for long operating times at high loads, the occurrence of a fault may require a forceful shutting down of the system for repairs, which may take a long time depending on the nature of the fault. This causes significant loss in production time, increased repair costs and in turn, lower profits. For a system working in such an environment, it would be advantageous if the potential occurrence of a fault can be detected beforehand rather than allowing it to occur and then reacting to it by performing a fault diagnosis. Hence, a recommended direction for future work is to extend the fault detection algorithm proposed in the current study to detect the faults in the system before they occur.
   In order to achieve this, the fault-detection algorithm must now perform a monitoring task where the operating states of the system are to be monitored over a certain time window. Based on that the algorithm will access the health of the system at the current time step and predicts whether the system shall continue to operate in a healthy state or if it is heading towards the occurrence of a fault in the near future. For the purpose of this monitoring task, long short-term memory (LSTM) neural networks would be suitable as they operate by providing a prediction for future states of the system for the next time-step 't+1' based on the states of the system that is observed at the current time-step 't', and the states of the system that were observed in the past time-steps 't-1', 't-2' and so on. Note that this is significantly different from the current fault-detection algorithm which classifies the system to be at fault based on the system's states at the current time-step without taking into consideration the history of the system's states.

3. **Extension to systems modelled using an unrestricted neural network architecture**
Recall that for a system that is modelled by a neural network containing a hidden layer in its architecture, the reconstruction of weights (fault identification) belonging to W_1 layer of this neural network requires the estimation of values of neurons in the hidden layers first. This estimation is carried out by solving a system of linear equations obtained using the neurons in hidden and output layers along with the weights lying in W_2 layer of the neural network, as was shown in figure 3.13. In this thesis a special case was considered where the architecture of the neural network modelling the system was such that the number of neurons in the hidden layer (2) was always equivalent to the number of neurons in the output layer (also 2), therefore resulting in obtaining 'n' number of equations (here 2) and 'n' number of unknown variables (here the 2 neurons in the hidden layer) to solve for i.e., a deterministic set of equations, solving which the accuracy with which the neurons of the hidden layer are estimated is quite high. This high accuracy of estimation in turn allows for a higher accuracy in the reconstructed weights when the SBL algorithms are applied.
However, this restriction placed on the architecture of the neural network limits the accuracy with which the neural network is able to model the system at hand. This is especially prominent in case of nonlinear systems where the training error obtained on training and test data sets are likely to be large as was the case in the previous chapter in experiment 3 in figures **??** and 4.14. This poor accuracy in modelling the system is then bound to affect the performance of the fault detection algorithm.
On the other hand, consider that the system is accurately modelled (low training error on the training and test data sets) by a neural network for which no restrictions are placed on it's architecture. Let this neural network have an unequal number of neurons in it's hidden layer and the output layer. Although by removing the restriction on the neural network architecture, the problem of accurately modelling the system in order for the fault-detection algorithm to perform well is taken care of, another problem is encountered: values of the neurons in the hidden layer can no longer be estimated accurately any longer. This is because a situation where the number of unknowns to be calculated (neurons in the hidden layer) now exceeds the number of known equations as was shown earlier in figure 3.14.
An attempt was made to reconstruct the weights in the W_1 layer of a neural network modelling a nonlinear system whose architecture had 6 neurons in the input layer, 3 neurons in the hidden layer and 2 neurons in the output layer. In this example it is assumed that it is already known that in total two changed-weights are present in the neural network modelling the nonlinear system:

   - First changed-weight in the W_1 layer specifically one among the set of weights going from first neuron in input layer to the three neurons in the hidden layer i.e., NN_1, NN_7 or NN_{13} (shown in blue in the table).
   - Second changed-weight in the W_2 layer specifically one among the set of weights going from the three neurons in hidden layer to the two neurons in the output layer i.e., NN_{19}, NN_{20} or NN_{21} (shown in red in the table).

   Results of the reconstruction can be seen in table 5.1. The neural network architecture is a 6-3-2 with 6 neurons in the input layer, 3 neurons in the hidden layer and 2 neurons in the output layer. It can be seen from the table that the accuracy of the reconstructed weights (highlighted in blue) belonging to the W_1 layer is poor. This poor reconstruction is a reflection of the accuracy with which the values of the neurons in the hidden layer were estimated by solving the pair of linear equations.
Hence a recommended direction for future work, is to propose a method for accurately estimating values of the neurons in the hidden layer of the neural network in a manner that does not place a restriction on the neural network architecture required to model the system.

4. **Intuitive method for identification of the possible locations of the changed-weights in the neural network modelling the system**

Layer in which the weight is: W1/W2	Neural network weight #	Magnitude of the trained weights	Magnitude of the weights after fault in system	Reconstructed weights from SBL algorithm
W1	NN 1	-0.6691	2.3309	2.3552
	NN 2	-0.9520	-0.9520	-0.9520
	NN 3	0.5670	0.5670	0.5670
	NN 4	-0.3272	-0.3272	-0.3272
	NN 5	0.2570	0.2570	0.2570
	NN 6	0.0158	0.0158	0.0158
	NN 7	-0.3193	-0.3193	-0.1900
	NN 8	0.8967	0.8967	0.8967
	NN 9	0.5845	0.5845	0.5845
	NN 10	0.7714	0.7714	0.7714
	NN 11	-0.6051	-0.6051	-0.6051
	NN 12	-0.6026	-0.6026	-0.6026
	NN 13	-0.3566	-0.3566	0.0970
	NN 14	-0.2808	-0.2808	-0.2808
	NN 15	-0.8694	-0.8694	-0.8694
	NN 16	0.1934	0.1934	0.1934
	NN 17	0.8466	0.8466	0.8466
	NN 18	1.0922	1.0922	1.0922
W2	NN 19	1.3965	16.3965	16.3284
	NN 20	9.5026	9.5026	9.5159
	NN 21	-8.6836	-8.6836	-8.6386
	NN 22	-9.3911	-9.3911	-9.3911
	NN 23	8.9410	8.9410	8.9410
	NN 24	-1.6618	-1.6618	-1.6618

Table 5.1: Results obtained using the SBL algorithm for the reconstruction of red and blue weights lying in the W_1 and W_2 layer.

Recall that the reconstruction of weights (fault identification task) in the W_1 and W_2 layers requires the estimation of the values of the neurons in the hidden layer. This estimation requires having an intuition on possible locations of the weights that have changed their values in the neural network (modelling the system) after the occurrence of a fault in the system. In this thesis the proposed method to help build this intuition is by training new neural networks on the prediction errors obtained at the output states of the system. The intention behind this was that, the values or patterns in the values of the weights in these newly trained neural networks will help build an understanding on which weights in the neural network (modelling the system) that are most responsible for the prediction errors received at the output states. Building this intuition helps narrow down the number of possible locations for the changed-weights from several to only a few in number.
Having said this, as was seen in the case of linear and nonlinear systems (experiments 2 and 3) modelled by neural networks having a hidden layer, identifying patterns in the values of the weights of these newly trained networks may not be so straightforward and rather complicated. Moreover these patterns varied for linear and nonlinear systems in these experiments. Not only this but the pattern will also vary for nonlinear systems that are modelled with different neural network architectures from one another for example: 6-2-2 vs 6-3-2 architecture.
While figuring out a pattern is time-consuming, it is still manageable for cases where the neural network architecture is relatively small in size. However in larger and more complex neural network architectures (possibly having multiple hidden layers in addition to several neurons in each layer) identifying a pattern in the values of the weights might be cumbersome or nearly impossible to do manually.
Hence a recommended direction for future work, is to extend the current method or propose an alternative method that makes it easy to identify the possible locations of the changed-weights in the neural network architecture modelling the system.

5. **Comparative study: Current fault diagnosis approach vs. fault diagnosis via system identification using basis functions**
The objective of the current work was to carry out a fault diagnosis of the system modelled using a neural network using the two proposed fault diagnosis algorithms. A possible direction for future work is to carry out a comparative study between the current fault diagnosis approach and a system identification based fault diagnosis approach using basis functions. These methods can be compared on the basis of factors such as the accuracy with which the fault diagnosis is carried out, complexity and prior requirements for implementation of either method and computational time.
A brief overview of the fault diagnosis of a system that has been modelled using basis function is explained as follows: Available measurement data corresponding to the system's normal operating conditions is used to identify an equation(s) of the system that represents the system's dynamics during normal operating condition. This is done by solving the SSR problem shown in figure 5.1. In this figure:

   - **y** is a '$N \times 1$' vector that represents the system's output for its single output state across the 'N' experiments/time-steps.
   - Φ is a dictionary matrix of dimension '$N \times M$' where each one of the M column represent a function. For example, assume that the M^{th} column of the dictionary matrix is the polynomial function x^3. The values of each entry in the M^{th} column of the dictionary matrix is given by the evaluating the polynomial function (x^3) across the 'N' experiments/time-series. The same will happen for each of the M-1 other columns.
   - **x** is the '$M \times 1$' vector that is to be solved for. This vector contains M coefficients corresponding to the M functions of the dictionary matrix. For example a value of '0' in the M^{th} row of vector x means that the function in the M^{th} column (here x^3) has a coefficient 0 in front of it which in turn implies that the x^3 term does not contribute to output 'y' of the system.

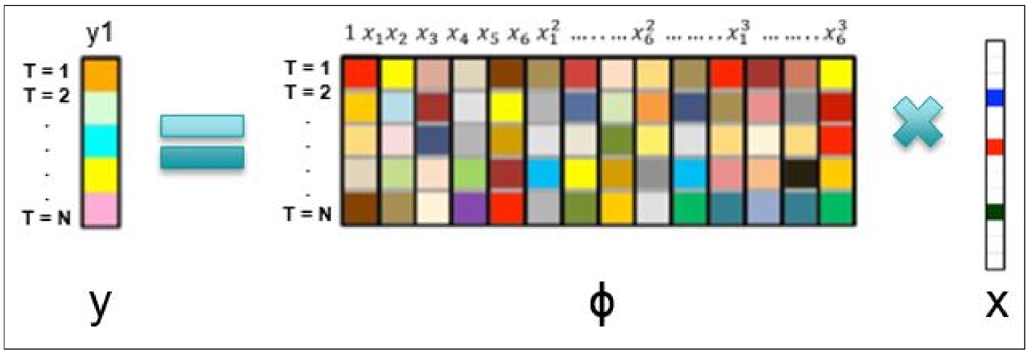

y1
T = 1
T = 2
T = N
1 x_1 x_2 x_3 x_4 x_5 x_6 x_1^2 .... x_6^2 .... x_1^3 .... x_6^3
y
ϕ
x

Figure 5.1: Identifying the equation of the system from the available measurement data using basis functions.

The fault-detection algorithm is identical to the one proposed in this work, i.e., a fault is said to have occurred in the system if the generated residuals (actual output - expected output from constructed model) fall outside the upper and lower threshold band as was shown in experiment 3 from the previous chapter in figure 4.17. Once the fault has been detected by the fault-detection algorithm, measurement data from the faulty system's input and output states are used to identify an equation(s) that represents dynamics of the faulty system. The fault identification task is then carried out on the assumption that by subtracting the equation(s) that represents the dynamics of the normal system from the equation(s) that represents the dynamics of the faulty system, an equation that models the fault in the system can be obtained i.e., fault in system = faulty system dynamics - normal system dynamics.

An example of this approach for carrying out a fault diagnosis is shown as follows:

consider the nonlinear system given by its true equation 5.1.

$$y = 0.5x_1^3 + 2x_2 + 1.1x_3^2 + 0.95x_4 - 0.5x_5 - 2x_6 \tag{5.1}$$

First the data from the normal operating conditions of the system is collected in order to identify an equation for the normal operating condition using basis functions. The equation obtained for the system is given by equation 5.2. It can be seen that the equations of the system found using the basis functions are highly accurate.

$$y = 0.4999x_1^3 + 1.9998x_2 + 1.1x_3 + 0.94977x_4 - 0.49976x_5 - 1.9997x_6 \tag{5.2}$$

Lets say that at a certain time-step 't' during operation, the system undergoes a fault as a result of which the faulty system dynamics operates as per equation 5.3.

$$y = 1.5x_1^3 + 2x_2^2 + 1.1x_3^2 + 0.95x_4 - 0.5x_5 - 2x_6 \tag{5.3}$$

Following the detection of the fault in the system by the fault-detection algorithm, measurement data from the faulty system's input and output states are collected and then used to identify an equation(s) that represents the dynamics of the faulty system, the obtained equation is given in equation 5.4.

$$y = 1.4999x_1^3 + 1.9999x_2^2 + 1.0998x_3^2 + 0.94843x_4 - 0.49858x_5 - 1.9986x_6 \tag{5.4}$$

After the equation for the system operating under fault has been obtained, the fault identification task is then carried out by subtracting the equation that represents the dynamics of the normal system from the the equation that represents the dynamics of the faulty system (fault in system = faulty system dynamics - normal system dynamics). The equation that models the fault in the system obtained is given in equation 5.5.

$$y_{error} = 0.99994x_1^3 + 1.9999x_2^2 - 1.9998x_2 \tag{5.5}$$

Bibliography

[1] David Henry, Silvio Simani, and Ron J Patton. Fault detection and diagnosis for aeronautic and aerospace missions. In *Fault tolerant flight control*, pages 91–128. Springer, 2010.

[2] Venelin Valkov. Neural network, 2017. [Online; accessed April 27, 2019].

[3] Sachin Joglekar. Neural network forward propagation, 2015.

[4] Akash Goel. Neural network backpropagation learning, 2016.

[5] Jakob Aungiers. point-by-point prediction, 2018.

[6] Deepthi Cheboli. Anomaly detection of time series. Master's thesis, UNIVERSITY OF MINNESOTA, 2010.

[7] B Samanta. Gear fault detection using artificial neural networks and support vector machines with genetic algorithms. *Mechanical systems and signal processing*, 18(3):625–644, 2004.

[8] Venkat Venkatasubramanian and King Chan. A neural network methodology for process fault diagnosis. *AIChE Journal*, 35(12):1993–2002, 1989.

[9] Joshua Owoyemi. Kalman filter, 2017.

[10] Bhaskar D Rao. Bayesian methods for sparse signal recovery.

[11] Achmad Widodo and Bo-Suk Yang. Application of nonlinear feature extraction and support vector machines for fault diagnosis of induction motors. *Expert Systems with Applications*, 33(1):241–250, 2007.

[12] Nassim Laouti, Nida Sheibat-Othman, and Sami Othman. Support vector machines for fault detection in wind turbines. *IFAC Proceedings Volumes*, 44(1):7067–7072, 2011.

[13] Sheng-Fa Yuan and Fu-Lei Chu. Support vector machines-based fault diagnosis for turbo-pump rotor. *Mechanical Systems and Signal Processing*, 20(4):939–952, 2006.

[14] Setu Madhavi Namburu, Mohammad S Azam, Jianhui Luo, Kihoon Choi, and Krishna R Pattipati. Data-driven modeling, fault diagnosis and optimal sensor selection for hvac chillers. *IEEE transactions on automation science and engineering*, 4(3):469–473, 2007.

[15] Riccardo Muradore and Paolo Fiorini. A pls-based statistical approach for fault detection and isolation of robotic manipulators. *IEEE Transactions on Industrial Electronics*, 59(8):3167–3175, 2012.

[16] Uwe Kruger and Grigorios Dimitriadis. Diagnosis of process faults in chemical systems using a local partial least squares approach. *AIChE Journal*, 54(10):2581–2596, 2008.

[17] Jong-Min Lee, ChangKyoo Yoo, Sang Wook Choi, Peter A Vanrolleghem, and In-Beum Lee. Nonlinear process monitoring using kernel principal component analysis. *Chemical engineering science*, 59(1):223–234, 2004.

[18] Leo H Chiang, Evan L Russell, and Richard D Braatz. Fault diagnosis in chemical processes using fisher discriminant analysis, discriminant partial least squares, and principal component analysis. *Chemometrics and intelligent laboratory systems*, 50(2):243–252, 2000.

[19] DC Baillie and J Mathew. A comparison of autoregressive modeling techniques for fault diagnosis of rolling element bearings. *Mechanical Systems and Signal Processing*, 10(1):1–17, 1996.

[20] RJ Patton and J Chen. Neural networks in fault diagnosis of nonlinear dynamic systems. *ENGINEERING SIMULATION C/C OF ELEKTRONNOE MODELIROVANIE*, 13:905–924, 1995.

[21] Majid Jamil, Sanjeev Kumar Sharma, and Rajveer Singh. Fault detection and classification in electrical power transmission system using artificial neural network. *SpringerPlus*, 4(1):334, 2015.

[22] Feng Jia, Yaguo Lei, Jing Lin, Xin Zhou, and Na Lu. Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data. *Mechanical Systems and Signal Processing*, 72:303–315, 2016.

[23] J Rafiee, F Arvani, A Harifi, and MH Sadeghi. Intelligent condition monitoring of a gearbox using artificial neural network. *Mechanical systems and signal processing*, 21(4):1746–1754, 2007.

[24] Hua Su and Kil To Chong. Induction machine condition monitoring using neural network modeling. *IEEE Transactions on Industrial Electronics*, 54(1):241–249, 2007.

[25] Juntong Qi, Xingang Zhao, Zhe Jiang, and Jianda Han. An adaptive threshold neural-network scheme for rotorcraft uav sensor failure diagnosis. In *International symposium on neural networks*, pages 589–596. Springer, 2007.

[26] Bin Jiang, Marcel Staroswiecki, and Vincent Cocquempot. Fault identification for a class of time-delay systems. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 3, pages 2239–2244. IEEE, 2002.

[27] Richard Washington. On-board real-time state and fault identification for rovers. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 1175–1181. IEEE, 2000.

[28] Ricardo Dunia, S Joe Qin, Thomas F Edgar, and Thomas J McAvoy. Use of principal component analysis for sensor fault identification. *Computers & Chemical Engineering*, 20:S713–S718, 1996.

[29] Gang Li, Carlos F Alcala, S Joe Qin, and Donghua Zhou. Generalized reconstruction-based contributions for output-relevant fault diagnosis with application to the tennessee eastman process. *IEEE transactions on control systems technology*, 19(5):1114–1127, 2011.

[30] Marcello R Napolitano, Younghwan An, and Brad A Seanor. A fault tolerant flight control system for sensor and actuator failures using neural networks. *Aircraft Design*, 3(2):103–128, 2000.

[31] Ihab Samy, Ian Postlethwaite, and Da-Wei Gu. Sensor fault detection and accommodation using neural networks with application to a non-linear unmanned air vehicle model. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 224(4):437–447, 2010.

[32] Lennart Ljung. Black-box models from input-output measurements. In *IMTC 2001. Proceedings of the 18th IEEE instrumentation and measurement technology conference. Rediscovering measurement in the age of informatics (Cat. No. 01CH 37188)*, volume 1, pages 138–146. IEEE, 2001.

[33] Ljung Lennart. System identification: theory for the user. *PTR Prentice Hall, Upper Saddle River, NJ*, pages 1–14, 1999.

[34] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[37] Julian D Olden and Donald A Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1-2):135–150, 2002.

[38] Orna Intrator and Nathan Intrator. Interpreting neural-network results: a simulation study. *Computational statistics & data analysis*, 37(3):373–393, 2001.

[39] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.

[40] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[41] BM Wise and NL Ricker. Recent advances in multivariate statistical process control: improving robustness and sensitivity. In *IFAC Symposium on Advanced Control of Chemical Processes. Toulouse, France*, pages 125–130. Citeseer, 1991.

[42] H Henry Yue and S Joe Qin. Reconstruction-based fault identification using a combined index. *Industrial & engineering chemistry research*, 40(20):4403–4414, 2001.

[43] S Joe Qin. Statistical process monitoring: basics and beyond. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 17(8-9):480–502, 2003.

[44] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[45] B Samanta, KR Al-Balushi, and SA Al-Araimi. Artificial neural networks and support vector machines with genetic algorithm for bearing fault detection. *Engineering applications of artificial intelligence*, 16(7-8):657–665, 2003.

[46] Richard Markert, Roland Platz, and Malte Seidler. Model based fault identification in rotor systems by least squares fitting. *International Journal of Rotating Machinery*, 7(5):311–321, 2001.

[47] Ricardo Dunia, S Joe Qin, Thomas F Edgar, and Thomas J McAvoy. Sensor fault identification and reconstruction using principal component analysis. *IFAC Proceedings Volumes*, 29(1):6578–6583, 1996.

[48] Ricardo Dunia and S Joe Qin. Subspace approach to multidimensional fault identification and reconstruction. *AICHE journal*, 44(8):1813–1831, 1998.

[49] Sang Wook Choi, Changkyu Lee, Jong-Min Lee, Jin Hyun Park, and In-Beum Lee. Fault detection and identification of nonlinear processes based on kernel pca. *Chemometrics and intelligent laboratory systems*, 75(1):55–67, 2005.

[50] Ji-Hoon Cho, Jong-Min Lee, Sang Wook Choi, Dongkwon Lee, and In-Beum Lee. Fault identification for process monitoring using kernel principal component analysis. *Chemical engineering science*, 60(1):279–288, 2005.

[51] Carlos F Alcala and S Joe Qin. Reconstruction-based contribution for process monitoring with kernel principal component analysis. *Industrial & Engineering Chemistry Research*, 49(17):7849–7857, 2010.

[52] Simon Foucart. Sparse recovery algorithms: sufficient conditions in terms of restricted isometry constants. In *Approximation Theory XIII: San Antonio 2010*, pages 65–77. Springer, 2012.

[53] Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.

[54] Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on information theory*, 53(12):4655–4666, 2007.

[55] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[56] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[57] Kenneth Lange. *MM optimization algorithms*, volume 147. SIAM, 2016.

[58] Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd. Enhancing sparsity by reweighted ℓ 1 minimization. *Journal of Fourier analysis and applications*, 14(5-6):877–905, 2008.

[59] Michael E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244, 2001.

[60] Wei Pan. *Bayesian learning for nonlinear system identification*. PhD thesis, Imperial College London, 2015.

[61] Zhilin Zhang, Tzyy-Ping Jung, Scott Makeig, and Bhaskar D Rao. Compressed sensing for energy-efficient wireless telemonitoring of noninvasive fetal ecg via block sparse bayesian learning. *IEEE Transactions on Biomedical Engineering*, 60(2):300–309, 2013.

[62] Youness Arjoune, Naima Kaabouch, Hassan El Ghazi, and Ahmed Tamtaoui. Compressive sensing: Performance comparison of sparse recovery algorithms. In *2017 IEEE 7th annual computing and communication workshop and conference (CCWC)*, pages 1–7. IEEE, 2017.

[63] David P Wipf. Sparse estimation with structured dictionaries. In *Advances in Neural Information Processing Systems*, pages 2016–2024, 2011.

[64] Krzysztof Patan, Marcin Witczak, and JóZef Korbicz. Towards robustness in neural network based fault diagnosis. *International Journal of Applied Mathematics and Computer Science*, 18(4):443–454, 2008.

[65] Chris Bishop. Exact calculation of the hessian matrix for the multilayer perceptron, 1992.