Issues in learning cause and effect relationships from examples: With particular emphasis on casting processes.

Ransing, Meghana R

SCHOOL OF ENGINEERING

UNIVERSITY OF WALES SWANSEA

ISSUES IN LEARNING CAUSE AND EFFECT RELATIONSHIPS FROM EXAMPLES: WITH PARTICULAR EMPHASIS ON CASTING PROCESSES

MEGHANA R. RANSING

B.ENG.(COMPUTER) (PUNE)

THESIS SUBMITTED TO THE UNIVERSITY OF WALES IN CANDIDATURE FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DECEMBER 2002

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree at any other university.

Signed ....(candidate)

Date 24|02|2003 ....

STATEMENT 1

This dissertation is being submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

Signed . ...(candidate)

Date 24|02|2003 ....

STATEMENT 2

This dissertation is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by giving explicit references.

Signed ....(candidate)

Date 24|02|2003 ....

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loans **after expiry of a bar on access approved by the University of Wales on the special recommendation of the Constituent Institution/University College concerned.**

Signed ... .(candidate)

Date 24|02|2003 ....

To my son, Kumar.

Summary

The aim of this research is to provide a self-learning, computer based, decision-making tool for industry. The tool will have a knowledge of current/past rejection levels within the manufacturing set up, the diagnosis done by experts and will use this information to automatically learn a cause and effect relationship.

The work presented in this thesis constitutes an algorithm for associating belief values in the occurrence of a cause (or a combination of causes) with corresponding belief values in the occurrence of an effect (or a combination of effects).

For this research, the neural network approach was first chosen because of its potential advantages over the traditional rule based approach. The influence of network parameters such as weights, biases, learning rate, momentum term and mode of training, has been analysed on network training. The analysis of the influence of the variation of a gain value during the training and testing phase showed that gain is not an independent parameter as perceived before, but depends on the initial weight values and the learning rate value. It was discovered that the variation in the gain value also influences the learning speed. A coupled algorithm has been proposed in this thesis to change the gain value adaptively.

It has been found during this research that neural networks are probably not the best available techniques for learning cause and effect relationships from examples, particularly due to their poor extrapolation abilities on incomplete and noisy training data sets. A novel and efficient method has been proposed, which overcomes the major limitation of the poor extrapolation ability of neural networks. This was achieved by effectively storing prior knowledge about the cause and effect relationship within the network. Enhancements have also been introduced to make this algorithm efficient. The belief value in the occurrence of the likely causes of one or more given effects is determined using this method.

The algorithm developed is generic and is applicable to all manufacturing processes and possibly in all situations where the cause and effect relationship is complex and a data set associating belief values in causes and effects is available.

Acknowledgements

Words are not enough to express my gratitude to Prof. R. W. Lewis, my PhD supervisor. It is because of him, today, I am enjoying a sense of achievement and satisfaction over my research output. He has instilled in me the confidence and an ability to take research decisions. He stood firmly behind me during those ups and downs, always giving me a helping hand to overcome my limitations.

My appreciation also goes to my husband, Rajesh, who was also my work colleague. Discussions with him were inspiring. He has also boosted my confidence with his friendship in times of both trial and jubilation.

Many thanks to those in the Casting Research Group and the fellow students for their company.

I would also like to acknowledge the financial support given to me through departmental assistantships.

Finally, my deepest gratitude goes to my parents and my sister, Nutan. Your constant encouragement and love has helped me through it all.

Contents

Glossary of Terms

Semantically Constrained Neural Network	The network is constructed by associating input nodes to defect nodes, hidden nodes to metacause nodes and output nodes to rootcause nodes.
Rootcauses	Design, process and material parameters which can be controlled directly.
Metacauses	An intermediate cause that governs the occurrence of defects and can be controlled by controlling one or more rootcauses.
Defects	Defect types.
Expert systems	These are based on a rule based architecture with backward chaining as a reasoning mechanism, have been developed for casting defect analysis.
weight	It is a numerical value assigned to every connection between two nodes of consecutive layers. It indicates the strength of the association between two nodes.
bias	It changes the value of the weighted sum at which the sigmoid function changes it's output value from zero to one. It is implemented by adding a new weight having a unit input value.
gain	It describes the slope of the sigmoid activation function.
net input	Net input activation which is the sum of weighted inputs to a node.
output	output signal

input nodes	Input to these nodes form an input to the network. These inputs are generally filtered before presenting to a neural network as input.
output nodes	outputs from these nodes represent the output of a network.
hidden nodes	these receive inputs from a layer of nodes and their output form an input to the nodes in the next layer of the network. Hidden nodes define the non-linearity of the decision surface.
layers	Nodes in a network are arranged in the form of layers.
Single Layer Neural Networks	Neural Networks having only one input layer and one output layer of neurons.
Multi Layer Neural Networks	Neural Networks having one input layer, one output layer and at least one hidden layer of neurons.
Logistic Sigmoid Activation Function	'S'-shaped function that generates output for a node in a predefined range of 0 to 1 as a function of the weighted sum.
Network Training	During this stage, all the network parameters are determined by presenting an example set of known input and output pairs and minimizing the error between network prediction and known output during each presentation. The optimal values of network parameters are then used for testing the network.
Training Data Set for Supervised Training	Examples with inputs and associated outputs.
Network Testing	The performance of the trained network is tested on another set of known input and output examples.
Testing Data Sets for Supervised Training	Similar to training data sets.
Supervised Training	Training process in which adjustments to network parameters are carried out iteratively in the presence of a supervisor, having the knowledge of the environment.

Back-Propagation Algorithm	Example of supervised training.
Sequential Mode of Learning	Weight updating is performed after presentation of each training example.
Batch Mode of Learning	Weight updating is done after presenting the entire set of training examples that constitute an epoch.
Learning Rate	Determines the magnitude of weight change along the chosen direction in weight space and therefore influences the training speed of a network.
Momentum Constant	A fraction which determines the proportion of weight change in the previous iteration that is added to the weight change in the current iteration. This improves learning speed.
Target Error	An acceptable low value over which the sum squared error over the entire training set is reduced.
Regression Coefficients	Independent variables which are generally determined using least square minimisation techniques.
Lagrange Interpolation Polynomials	Linear, quadratic, cubic or higher ordered polynomials which have some particular attributes. 1. At a given time, the value of the polynomial at a node is equal to unity and zero at all other nodes. 2. Sum of all values of polynomials equals unity.
Reference Points	For an n^{th} order relationship there are $(n+1)$ equidistant reference points ranging from -1 to $+1$ along a dimension in the input space created by the strength of effects of a cause.
Primary Reference Points	Reference points along the axes.

Secondary Reference Points	All other reference points.
Primary Weight Values	Weight values associated with primary reference points are the independent parameters in the network.
Secondary Weight Values	Weight values associated with secondary reference points are linear combination of the corresponding primary weight values.
Optimisation Algorithm	Gradient Descent, Quasi-Newton, Levenberg Marquardt are the algorithms used for this thesis.
Gaussian Noise	Normally distributed noise using standard deviation as 20% throughout this thesis.
Extrapolation Ability	Predicting the value of unknown data points by projecting a function beyond the range of known data points.

Nomenclature

$i \quad (i = 1\,to\,l)$	i^{th} input node and l is the total number of input nodes.
$j \quad (j = 1\,to\,m)$	j^{th} hidden node in the first hidden layer and m is the total number of hidden nodes in that layer
$k \quad (k = 1\,to\,n)$	k^{th} output node in the output layer and n is the total number of output nodes
o_i	Output signal of i^{th} unit
$o_1, \ldots o_i, \ldots, o_l$	Set of input signals (suffice i and l are used for input nodes)
$x_1, \ldots x_i, \ldots, x_l$	Set of input signals
$h_1, \ldots h_j, \ldots, h_m$	Set of output signals of hidden nodes in the first hidden layer (suffice j and m are used for hidden nodes)
$o_1, \ldots o_j, \ldots, o_m$	Set of output signals of hidden nodes in the first hidden layer
$o_1, \ldots o_k, \ldots, o_n$	Set of output signals of output nodes in the output layer (suffice k and n are used for output nodes)
w_{ij}	Weight of the link from unit i to unit j
w_{jk}	Weight on the link from unit j to unit k
$a_{net,j}$	Net input activation for the j^{th} unit
a_j	Activation value of the j^{th} unit

o_j	Output value of the j^{th} unit
o_k	Network output value of the k^{th} output unit
θ_j, w_0, b_0, h_0	Bias for the j^{th} unit
c_j	Gain of the j^{th} unit
E	Root Means Square Error signal
t_k	Desired output of the k^{th} output unit
η	Learning rate value
$\partial \bullet$	Partial derivative of $\bullet$
$\Delta \bullet (n)$	Weight, bias or gain correction term of $\bullet$ for n^{th} training iteration
μ	Momentum term value
$c \quad (c = 0\, to\, 1)$	Output belief value in a cause
$\xi^j \quad (j = 1\, to\, p)$	Belief values for 'p' effects
$z_i \quad (i = 1\, to\, m)$	Function of $\xi^j \quad (j = 1\, to\, p)$
$w_j\ (j = 0\, to\, p)$	Regression coefficients (in a regression analysis context) or weights (in a neural network context)
X1	Belief value in the occurrence of effect ξ^1
X2	Belief value in the occurrence of effect ξ^2
Output Value	Belief value in the occurrence of the associated cause c
n_j	Order of the Lagrange Interpolation Polynomial (one for linear, two or quadratic, etc.) corresponding to the j^{th} dimension
$i \quad (i = 1\, to\, n+1)$	For a 'p' dimensional case, 'i' ranges from one to total number of reference points 'q'. One-dimensional Lagrange Interpolation Polynomial is constructed for each reference point along each dimension.
$\xi_0^j, \xi_1^j, \xi_2^j, \ldots, \xi_{n_j}^j$	Coordinates of $(n_j + 1)$equidistant reference points from $\xi_0 = -1$ to $\xi_n = +1$

$(k_1, k_2, \ldots, k_j, \ldots, k_p)$	The co-ordinates for a reference point 'i', corresponding to each dimension
q	Total number of reference points
w_i	Weight variable associated with the i^{th} reference point
C	Coefficient used in the linear combination expression.

List of Figures

List of Tables

Chapter 1

Introduction

This chapter contains an overview of the work background, scope and research contributions made. In addition to this it also gives an outline of the thesis.

1.1 Background

In recent times, various disciplines, including the manufacturing industry and at least some branches of the medical field, have come under increasing pressure to improve yield, productivity and profits (or reduce costs and overheads). As such, there is an increasing need to reduce costs and reach the desired result as quickly as possible, for example, in the manufacturing industry, the required result may be the manufacture of a batch of products which are all of optimum quality, and in a branch of the medical field, the required result might be the correct diagnosis of a complaint or disease in a patient (i.e. in both cases, to get it "*right first time*").

The cause and effect relationship is complex for many manufacturing processes and in most cases '*experience*' is the only factor which can help to take corrective actions. In the case of the manufacturing industry, manufactured products are usually tested for quality, and sub-standard components are rejected. When a component or set of components is rejected, the fault or faults are noted and reasons for the occurrence of the faults are established and corrective actions are taken. In this way, the chances of manufacturing sub-standard components thereafter are minimised. Such a diagnosis is usually performed by experts in the field, who have acquired a fundamental understanding of the process over years of experience in analysing cause and effect relationships. This is a time consuming process and furthermore, when an expert leaves a particular industry, his expertise is also lost to that employer.

The ability to learn causal relationships from diagnostic examples is extremely useful. The aim of this research is to provide industry with a **self learning** decision making tool, which has the knowledge of current/past rejection levels within the manufacturing set up, the diagnosis done by experts and which can use this information to automatically learn a cause and effect relationship. This learning ability will help managers not only to quantify the influence of causes on defects for existing components but also to set up new processes, material and design parameters to manufacture new components. Such a tool will also help industry to retain some of the expertise when experienced staff either retire or leave the job.

Neural network technology represents a meaningfully different approach to using computers in a workplace. A neural network is used to learn patterns and relationships in data. Having inter-relationships in the data means that two or more factors work together to predict the model outcome. The data may be the result of a market research effort, the result of a production process given varying operational conditions, or the study of cause and effect relationships for different situations from health services, manufacture of engineering components to various social management issues. For example, in the manufacture of engineering components using a casting process, shrinkage related defects might be caused by various reasons such as, incorrect number and position of feeders, chills and insulation, pouring temperature of the melt etc. Often, one or more factors combine to develop hot spots that may lead to shrinkage defects. Neural networks can be trained to predict whether hot spots are likely to be formed for given process conditions. They have generally excelled when an unknown non-linear relationship exists between explanatory factors (e.g. defects or input data) and the outcome (causes or output data). Neural networks discover this non-linear relationship during a training phase when the input and output data are repeatedly presented to the network. The output data is compared with the results calculated by the neural network and the difference, or the error, is processed via a mathematical procedure, which adjusts the value of network parameters (such as weights, biases etc) in order to minimise this error. However, neural networks have a few limitations which were discovered during this research, and because of these it is difficult to use this technique as a robust tool by end users in a foundry environment or any other manufacturing industry to analyse cause and effect relationships.

This research focuses on three major issues regarding cause and effect relationships.

- Quantifying the influence of one cause (or a combination of causes) on the occurrence of one effect (or a combination of effects) by studying past diagnostic examples.
- For analysing a cause and effect relationship, a fundamental understanding of the variation of belief values in the occurrence of a cause, given a set of symptoms or effects, is necessary. This research focuses on characterising this belief variation.
- Developing a robust algorithm that will take advantage of the 'learning from example' paradigm of neural networks and at the same time overcoming the inherent problems associated with this technique.

1.2 Scope of Work and Research Contributions

The main research activities, in order to develop a robust tool for analysing 'cause and effect' relationships, were as follows:

- Define the diagnostic problem so that computer based models can be effectively developed.
- Discover some of the practical limitations in generating training data sets in a manufacturing environment.

- Compare two approaches, the first being the most widely used rule based approach, and the second approach i.e. the network based approach to analyse 'cause and effect' relationships.
- Compare the defect-metacause-rootcause type of representations for the causal knowledge over rule-based representaion of causal relationships.
- Study the applicability of neural network techniques for the diagnostic problem and also analyse the effect of neural network parameters on the performance of training using the back propagation algorithm.
- Compare neural network models with regression analysis techniques in order to find out the similarities and dissimilarities between both approaches.
- Develop a robust tool for quantifying cause and effect relationships by retaining the advantages of neural network and regression analysis techniques and overcoming their limitations.

Research contributions made during the above activities are summarised as follows:

- The "*cause and effect*" relationship for casting processes is a complex subject. The rejection data for a given casting component and time frame, normally indicates a pattern, which has a few defects occurring at significantly high proportions and some occurring at significantly low proportions. The diagnostic problem, therefore, was defined as recognising the patterns in the rejection data, and correspondingly, associating those patterns with a combination of causes. The belief values in the occurrence of these effects are associated with the corresponding belief values in the occurrence of the causes based on a known set of training examples.
- A network based approach was chosen due to its potential advantages over the rule based approach for automatically quantifying cause and effect relationships using past diagnostic examples.
- Most of the application oriented papers on neural networks have implied that neural networks operate like a "magic black box", which can simulate the "learning from example" ability of our brain with the help of network parameters such as weights, biases, gain, hidden nodes etc. There are very few publications, or textbooks, which give a physical interpretation for various

parameters used in the network, and in particular, most of the publications in the literature recommend a unit value for gain. The influence of the variation of gain value during training and testing phase has been analysed. Equations for weight, bias and gain updation were derived. It was discovered that for the back propagation algorithm, gain is not an independent parameter as perceived before, but depends on the initial weight values and learning rate value. A variation in the gain value results in faster training. A coupled algorithm has been proposed for the first time to adaptively adjust the learning speed of each node within the network.

- It has been recommended that neural networks, or Semantically Constrained neural networks, are not suitable for learning cause and effect relationships from examples because of their poor extrapolation abilities on incomplete and noisy training data sets.
- A novel and efficient method has been proposed for the first time, which uses the advantages of both neural network and regression analysis techniques while overcoming their limitations. This method overcomes the major limitation of the poor extrapolation ability of neural networks by effectively storing prior knowledge about cause and effect relationship within the network. Enhancements have also been introduced to make this algorithm efficient. The belief value in the occurrence of likely causes of one or more given effects is determined using this method.
- An international patent application (PCT / GB2002 / 003805) has been made to protect this invention. On the advice of UWS Ventures Ltd, a wholly owned subsidiary of the University of Wales Swansea, which is responsible for protecting its intellectual property, the contents of this thesis are to be treated in strict confidentiality.
- Recognising the academic potential of this work, EPSRC has also awarded a three-year research funding in collaboration with Rolls Royce Plc to further develop this innovation.

1.3 Outline of the Thesis

The thesis is subdivided into seven chapters, including the introduction and conclusion chapter. The following is a summary of each.

- *Chapter Two. Computer Aided Defect Analysis:* In this chapter, two approaches have been explored for undertaking the diagnosis of defective castings: The rule-based approach and the neural network based approach. The diagnostic problem is defined and research goals are identified for this thesis. The chapter also summarises previous work done in this area.

- *Chapter Three. Introduction to Neural Networks:* The basic structure and function of neural networks has been introduced in this chapter. The back propagation algorithm - one of the most widely used algorithms for neural network training has been described and the effect of various network parameters (such as weights, biases, gain, learning rate, momentum term, sequential and batch modes of network training) on the performance of the network have been studied in detail. Weight and bias update expressions were derived.

- *Chapter Four. Effect of Gain on the Learning Abilities of Neural Networks:* The slope of the activation function determines the range over which the output values of a neural network are changed from zero to one for a given set of weighted inputs. In a feed forward neural network algorithm, this slope is directly influenced by a parameter referred to as 'gain'. In this chapter, the influence of the variation of 'gain' on the learning abilities of a single and multi layer neural network is analysed. Also, the influence of adaptive gain variation is analysed and equations have been developed to adaptively select a gain value for each node during each epoch. The results of the simulation are also discussed for sequential as well as batch modes of training.

- *Chapter Five. Neural Networks as a Regression Analysis Tool:* In this chapter, neural network based models are related with regression analysis models, and their advantages and limitations for analysing cause and effect relationships are compared.

- *Chapter Six. New Algorithm for Causal Analysis using Multi-Dimensional Lagrange Interpolation Polynomials:* An efficient algorithm has been presented in this chapter, which retains the advantages of neural networks and overcomes its limitations in learning the input-output mapping function in the presence of noisy, limited and sparse data. The belief values in the occurrence of likely causes of one or more given effects are determined using this method. Sample calculations of the proposed algorithm have been compared with neural network solutions for four representative cases.

- *Chapter Seven. Conclusions and Future Work:* The original research contributions are summarized and recommendations made for the continuation of the work through future research.

CHAPTER 2

Computer Aided Defect Analysis

The diagnosis of defective castings has always been a center of attention in the manufacturing as well as the research community. Based on a knowledge representation scheme, most diagnostic systems are classified into two categories. The first, perhaps the most widely used, is a rule-based approach. This computer-aided technique, which aroused some interest in the late eighties and early nineties, is known as an 'expert system' which is also referred to as knowledge based systems [1-5]. The second approach i.e. the network-based approach, has received greater attention. Recently, Ransing and Lewis [6-13] have proposed a diagnostic algorithm based on the defect-metacause-rootcause type of representation for the causal knowledge. Such a network based causal representation has certain definitive advantages over rule-based representation of causal relationships. These approaches have been discussed in this chapter so as to define the diagnostic problem and identify research goals for this thesis.

2.1 Introduction

In any Intelligent Diagnostic System, the causal representation determines the ability of the system to effectively diagnose and also justify its diagnosis. In casting processes, most of the expertise has been gained, over a period of years, by trial and error. In the late seventies the focus of research for improving die casting quality was greatly on the experimental side. Statistical techniques such as design of experiments, factorial analyses, and statistical quality control techniques were used to analyse the relationships between causes and defects [14-16]. The influence of design, process and material parameters on the quality of castings was correlated with these analyses. With ever increasing pressure to reduce design cycle time, traditional analysis techniques need constant research and improvement. The interest in developing intelligent diagnostic systems for manufacturing process such as casting is ever increasing. Computational tools are used to assist with the design of components and manufacturing processes. Many computer-aided techniques have emerged to improve the casting quality. In this chapter, the author explores the applicability of computer methods to assist a process engineer in decision making to either improve process design or to solve a manufacturing problem.

The aim of this work is to develop a set of algorithms, which would impart a decision making ability in computer programs for the following specialised topics.

1. On-line diagnosis

2. Identify potential trouble making parameters for new prototypes based on the current rejection trends.

3. Like humans, learn from past mistakes, success stories and other examples and also explain or justify results.

4. Perform the computer simulation of a casting process. If a defect is predicted then automatically improve the design until the casting is simulated as being defect free.

Clearly, the objective is to extend the current abilities of computer programs a step further towards intelligent decision making. In the late eighties, expert systems were expected to achieve this task. Apparently, the initial enthusiasm for expert systems research seems to have subsided. With rule-based systems, the success lies in the formulation of rules. For a causal analysis, the author feels that a rule-based approach has many limitations. Also, it can be argued that expert systems do not give any new information to an expert user. Therefore, it was decided to investigate other algorithms, which would use techniques such as neural networks i.e. traditional and semantically constrained neural networks and provide new information, even to an expert user.

This chapter takes a fresh look at various diagnostic paradigms and argues that network based paradigms can offer a better tool to aid decision-making at the production level. Section 2 summarises the approach used for defect analysis and other diagnostic approaches reported in the literature along with the diagnostic network topology known as the Semantically Constrained Neural Network. Thus, network topology has been used as a starting point for this research. Section 3 summarises the causal representation in a defect-metacause-rootcause form with sample examples. Finally, the last section refines the objective of this research.

2.2 Proposed Approach for Defect Analysis

In a majority of foundry plants, the data available on the number of castings poured, along with the number of castings being considered as accepted or rejected as defectives before and after machining, is usually carefully recorded. This data is encoded for each type of defect, for each day, week and month of a manufacturing campaign. This information is available for all casting components.

An assessment of a large number of data sets for defective castings for different time spans, revealed that a large proportion (50 to 80%) of all defective castings produced contain two or three types of defects. Also, a number of defect types are either not found at all, or in very small numbers. In other words, a number of defects are conspicuous by their absence or contribute to only a tiny proportion of defectives in the manufacturing campaign. However, this information is also considered as significant, mainly because of the close interactions between causes and defects. The non-occurrence of particular defects actually helps in eliminating the possibility of the occurrence of some causes that are also common to the defects occurring at significant proportions.

Thus, it can be stated that the rejection data for a given casting and time frame, normally indicates a pattern, which has normally few defects occurring at significantly high proportions and some occurring at significantly low proportions.

The diagnostic problem, therefore, was defined as recognising patterns in the rejection data and identifying a corresponding combination of causes. In a casting process, the causal relationship is highly inter-linked. It is observed that a combination of defects generally occurs as a result of a combination of causes. The following observation may be made.

A certain combination of defects generally occurs as a result of a combination of assignable causes. The reduction in the belief value in an assignable cause, due to the non-occurrence or the partial occurrence of a related defect, depends on –

1. The relative occurrence of a defect w.r.t. other defects and
2. The assignable cause under consideration.

The following sub-section explores various diagnostic tools available in the literature and analyses them in a way that could implement the proposed approach for the defect analysis.

2.2.1 Rule Based Approach

The casting process, being rich in experience and expertise, is a suitable vehicle for expert system application. Rule based, deterministic expert systems associated with backward chaining as a reasoning mechanism, have been developed for casting defect analysis [1, 5]. In these intelligent casting defect analysis applications, the handling of uncertainty [4] in the rules is considered through certainty factors as used in MYCIN [17] - one of the earlier and commercially successful expert systems. Many other expert systems were developed for casting analysis [1-5].

The rule based expert systems currently developed are essentially based on a rule based architecture [18] and either recognise a single defect or diagnose the causes of a single defect. The structure of rules in such expert systems follows a particular form that is represented as:

IF Condition 1 and Condition 2 and Condition3...
AND Symptom 1 and Symptom 2 and Symptom 3...
THEN Cause 1 (x%)
Where, x is the certainty factor.

Condition1, Condition 2 etc. could be the operating conditions such as pouring temperature range, the composition of certain constituents in the metal or the type of melting furnace used etc. The main advantage of this representation, apart from its simplicity, is that the diagnostic system can explain the potential outcome. In other words, the inference mechanism and knowledge storage is explicit and transparent to the user, and hence, it is easy to explain its logic. It is quite convenient to use expert system shells for developing such applications where the reasoning process, i.e., programs for handling uncertainty have already been developed and the knowledge engineer only needs to formulate appropriate rules to capture the expertise in a particular field.

In manufacturing processes such as casting, injection moulding or even machining processes, such as auto-turning, a large number of defects (ranging from 10 to 20) can occur in significantly high proportions as a result of various combinations of prevailing process conditions. Also, a number of defects can occur in significantly low proportions. Generally, the causal relationship is highly complex and inter-linked i.e. each cause influences a number of defects and each defect is influenced by a large number of causes. A comprehensive rule base would then require a large number of rules leading to other computational problems such as efficiency of the computation, consistency in the rules etc. This can be better explained by an example. Figure 2.1 shows a cause and effect relationship between two defects A, B and three causes C, D, E.

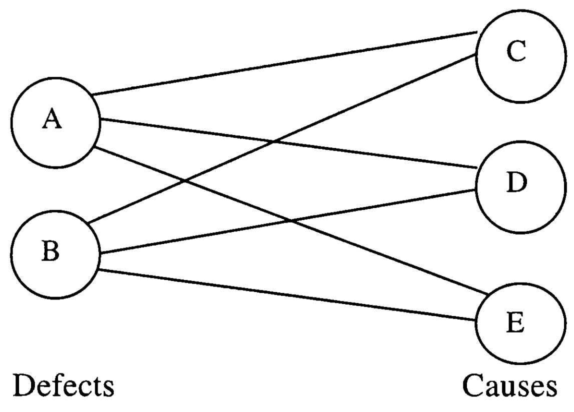


Figure 2.1 Complex cause and effect relationship.

For the example shown in Figure 2.1, the rule-based system would generate a large number of rules, because of the following reason. The defect A may occur if any of the causes i.e. C, D or E are occurring. Causes C, D and E may also occur together in any combination, and, every combination of causes may influence the occurrence of every combination of defects. The degree of influence of each cause (or a combination of causes) on the occurrence of each defect (or a combination of defects) is also different and this information needs to be stored in the rule base.

To overcome the problem of generating a large set of rules, a hypothetical function may be imagined in a manner that would take care of all possible permutations and combinations of causes and defects with a fraction indicating their degree of occurrences. The structure of this rule may be represented as:

IF C[c] & D[d] & E[e]

$$\left(A\ \textit{function}\ (F)\ \textit{which will automatically generate values 'a' and 'b' based on 'c', 'd' and 'e'}\right)$$

THEN A[a] & B[b]

Where a, b, c, d, e are fractions between 0 and 1, which correspondingly indicate the non-occurrence and occurrence for defects A, B and causes C, D and E.

In a rule based architecture, with a backward chaining reasoning mechanism, a number of questions needs answering before diagnosing each defect. The handling of uncertainty can be done through certainty factors. However, the creation of such a generalised function using a rule-based approach requires the knowledge of the degree of influence of the related cause (and/or a combination of causes) on the occurrence of each defect (and/or a combination of defects). In the authors' experience, generating such a probability distribution for the entire relationship is extremely difficult, if not impossible. Another practical problem is that it is very difficult, even for manufacturing personnel, to precisely quantify the causal relationship. This is because of the fact that different experts, depending upon their knowledge and experience, give different opinions on the degree of influence of a cause (and/or a combination of causes) on the occurrence of a defect (and/or a combination of defects). The causes of defects interact with each other in unclear and fuzzy ways. The crisp quantification of a relationship, via probability values or certainty factors, is very difficult.

The objective of this thesis is to generate the proposed function F that satisfies the following requirements.

1. The function should not require the probability distribution as input to the system.
2. However, based on past examples, it should automatically extract the probability distribution.
3. The function is generated and changed adaptively for every new example. It should then continue to correct itself for every new example.
4. It should also predict the correct output for any new example.
5. Last but not least, the function should also account for the occurrences, non-occurrences and partial occurrences of defects as well as causes for each example.

Unfortunately, the rule-based approach is unable to adapt or learn from past trends or examples. Therefore, it cannot generate such a function F. Hence, the rule-based expert systems are unable to implement the approach for defect analysis explained in the previous section.

A neural network, which is a network-based approach, has certain advantages over the rule-based approach. The main advantage of a neural network is that it can adapt and learn from past examples and then be used to quantify highly inter-linked relationships. Therefore, this approach has been explored for the diagnostic problem. The following sub-section introduces the network-based approach.

2.2.2 Neural Networks - A Network Based Approach

A network is developed by connecting layers of nodes with linkages in a particular way. Generally, the concepts or facts are associated with these nodes and the relations between them are categorised with the help of linkages. Neural Networks is an example of network based systems.

For a casting process, a foundry manufactures daily a large number of castings and, provides a solution to manufacturing process problems, or studies any rejections that might have occurred. In other words, valuable information is generated within the foundry every time a casting is poured. Researchers have attempted to exploit this information via neural network techniques. Recently, neural network technology has gained more popularity because, unlike the rule-based approach this technology offers a convenient computational tool that can adapt, learn and then also be used to quantify complex and highly inter-linked causal relationships. This ability makes the field of diagnosis a potential application for neural networks. The diagnostic knowledge, which the neural network learns, is stored in terms of weights, i.e. numerical values associated with the links connecting the network nodes and hence, is one of the most important variables in a network. A weight represents the strength of the association between network nodes.

Smith and her co-investigators [21-22] have done considerable research in this direction. Smith et. al. have shown that the back-propagation neural network can be successfully applied for quality control applications. Neural networks were also used for the diagnosis of hydraulic forging presses [23]. Martinez et. al. [24] have investigated its application to relate process conditions to the probable quality rating of casting. This predictive analysis was done for a slip casting process. Spelt et. al. [25] have used neural networks for classifying power plant sensor data and coupled this with an expert system for diagnostic purposes. Zhang and Huang have presented a state-of-the-art survey of neural network applications in manufacturing. These include applications of neural networks for engineering design, process planning, in solving scheduling problems, process modelling and control, in monitoring and diagnosis and quality assurance [27].

The semantically constrained neural network has a modified three layered 'feedforward' network architecture, proposed by Ransing and Lewis [13]. Generally, a traditional neural network consists of layers of nodes, with each node in a layer being connected to every node in the next layer. In a semantically constrained neural network, the network topology is constrained to a defect-metacause-rootcause relationship, and thereby, the connectivity of the nodes becomes constrained with defect nodes to form an input layer, metacause nodes to form a hidden layer and rootcause nodes to form an output layer. A rootcause is a design, process or material parameter, which may be controlled to minimize the occurrence of defective castings. A metacause is defined as a scientific rationale that governs the occurrence of defects and is influenced by controlling one or more rootcauses. The constrained connectivity differs from that of the traditional neural network. In this new network, a node is connected to a node in the next layer only if a causal relationship exists.

For the benefit of readers who are unfamiliar with these concepts, metacauses and rootcauses are briefly explained later in this chapter. The limitations of using a semantically constrained neural network have been outlined in Chapter 5 of this thesis.

2.3 Traditional Representation of Causal Relationship

The most common way of representing a causal relationship is via a "*cause and effect*" diagram, which is a list of causes for a particular defect. For a pressure die casting process, the cause and effect relationship for a defect "*gas holes*" is illustrated in Figure 2.1. This process involves the injection of molten metal into a die cavity under pressure. The metal cools inside the die thereby achieving the desired geometry within a given tolerance. The causes for defects range from process, material to design parameters.

Such a list can be generated via 'brain storming' sessions involving production personnel and machine operators and quality control staff. Other sources of information are also available such as handbooks, scientific papers etc. It is also important to first characterise a defect nomenclature followed in a particular foundry as the 'shop floor' names for casting defects could change from one foundry to other. For example, a production supervisor might identify a particular defect as *Incomplete Fill*, whereas, an engineer might refer to this as a *misrun* or a *mismake*. Webster [26] has used defect images in the expert system in order to enable the user to identify the defects whilst using the expert system. A pictorial representation of defects is always a better way. Currently, this facility has not been incorporated into the code, as the main objective was to analyse a defect and cause relationship. In short, generating a simple cause and effect diagram itself is a time consuming task and needs to be done carefully which involves participation from all levels.

2.3.1 Re-analysis of the Causal Representation

Once the cause and effect diagram is generated for all defects, the next task is more intellectual and demanding i.e. the interpretation and compilation of these "*cause and effect*" diagrams. This is important because a single cause can influence the occurrence of a large number of defects e.g. the gate velocity in a pressure die casting process influences the occurrence of *porosity*, *mismake*, *shrinkage and hot cracks*. The next task is to identify the underlying metacauses. The concept of metacauses has been developed in trying to answer the question such as why and how a cause influences the occurrence of a defect? For example, consider a defect *mismake*. *Mismakes* are in general an incomplete formation of the casting shape. These are usually in the form of small holes or pieces missing from casting sections.

A number of causes can influence the occurrence of *mismakes* e.g. *gate design, die temperature, metal temperature, filling time or fluctuation in nitrogen pressure* to name but a few. If we ask a question, how does the *fluctuation in nitrogen pressure* influence the occurrence of *mismakes*? Nitrogen pressure controls the injection pressure.

Therefore, any fluctuations might lead to the *loss of pressure during the solidification* of melt. This will obviously lead to an incomplete formation of the casting shape producing a defective casting with a *mismake*. There are other causes which can also trigger *the loss of pressure during solidification* of melt such as *slug length – less, fast shot set point, pressure rise time during second stage* or *gate thickness – less* to name but a few. A foundry engineer would agree that once there is a *loss of pressure during solidification*, triggered by whichever cause, it is likely that defects such as *mismakes, cold shuts or dimensional inaccuracy will occur*. We define the *loss of pressure during the solidification* as a metacause. Metacauses are the scientific rationale, which capture the physics behind the occurrence of defects. The second category of causes is defined as rootcauses. Rootcauses are the process, design and material parameters which influence the occurrence of metacauses e.g. *gate design, die temperature, metal temperature, filling time* etc. unlike metacauses, rootcauses are those causes, which can be controlled or altered directly.

Such a representation of causes in a *defect-metacause-rootcause* form is generic. For example, in a sand casting process one of the metacauses can be identified as *Gas pressure > metal pressure*. This means that during pouring and the onset of solidification the gases generated in the mould and/or the entrapped air are unable to escape. This causes an increase in the back pressure of the molten metal, which can lead to the formation of defects such as *gas defect, misrun* and *cold shuts, scars seams and plates* etc. In this manner, each of the metacauses gives a scientific rationale for the causal relationships. Among the rootcauses, the process parameters could involve any parameter that increases the moisture, or wetness, in the sand mould. e.g. *improperly dried cores, broken or disturbed core wash, high moisture in the sand* or *improper mulling/mixing* whereas the design parameters will involve design decisions such as inadequate venting, number of gates and their locations etc.

For the example shown in Figure 2.1, the process parameter *slow approach* is shown related to the formation of *gas holes*. The term *slow approach* is related to the plunger velocity. If the 'slow approach' is too short, then turbulence occurs within the melt in the feed system, which in turn leads to the entrainment of gases in the liquid metal giving rise to *gas holes*. Similarly, all other subsequent causes resulting in *gas holes* were analysed and five independent scientific rationales were identified which governed the occurrence of *gas hole* defects. The occurrence of any one of these scientific rationales can result in the formation of the defect *gas holes*. All the process and design parameters, which can influence the scientific rationales, were listed. The expression of knowledge in such a form not only gives a better understanding of the process but also enables an efficient computer aided diagnosis to be made. The expansion of a defect *gas hole* represented in this heirarchical form is shown in Figure 2.2.

Consider another example, where an improper investment shell mould build can cause a *pin hole defect* as shown in Figure 2.3. However, a scientific analysis of the relationships would reveal that improper shell build might induce *too low a dip weight* which may cause a *pin hole defect.* There are a number of process design and material parameters, which may induce *too low a dip weight.* It is therefore likely that the *pin hole* defect will occur if there is *too low a dip weight* irrespective of which process, design or material parameter has caused it. Such a classification makes the causal relationship more transparent and provides an insight into the physics of the underlying relationship. Figure 2.4 shows the defect-metacause-rootcause relationship for the cause and effect relationship of Figure 2.3.

The interconnection among the defects, metacauses and rootcauses are represented by a three tier structured graph similar to that shown in Figure 2.2 or Figure 2.4. It can be observed that rootcause – metacause – defect relationships are highly inter-linked. This implies that one defect is associated with more than one metacause, which itself is associated with more than one rootcause. Also, a defect and one rootcause is associated with more than one metacause.

2.3.2 Neural Networks Based on Defect-Metacause-Rootcause Relationship [13]

In a semantically constrained neural network, the connectivity among the network nodes is constrained as per the defect-metacause-rootcause diagram. This knowledge representation scheme matches very closely with the architecture of a three-layered feed forward neural network. The network is constructed by associating input nodes to defect nodes, hidden nodes to metacause nodes and output nodes to rootcause nodes. The nodes are connected to the nodes in the next layer only if a causal relationship exists. This is possible because the concept of a metacause is assigned to every hidden node.

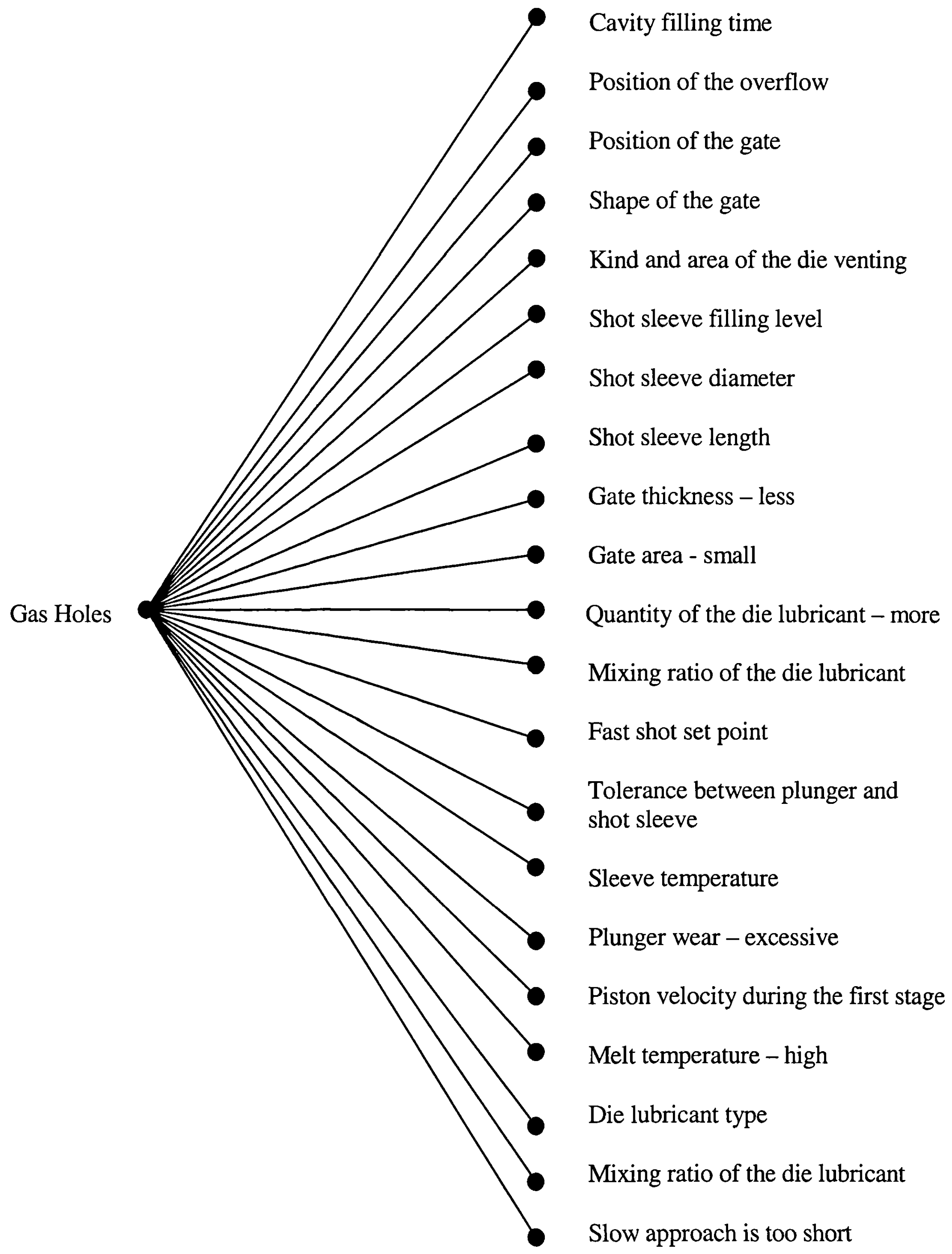


Figure 2.1: Cause and Effect Diagram for a defect *"Gas Hole"* for a Pressure Die Casting Process.

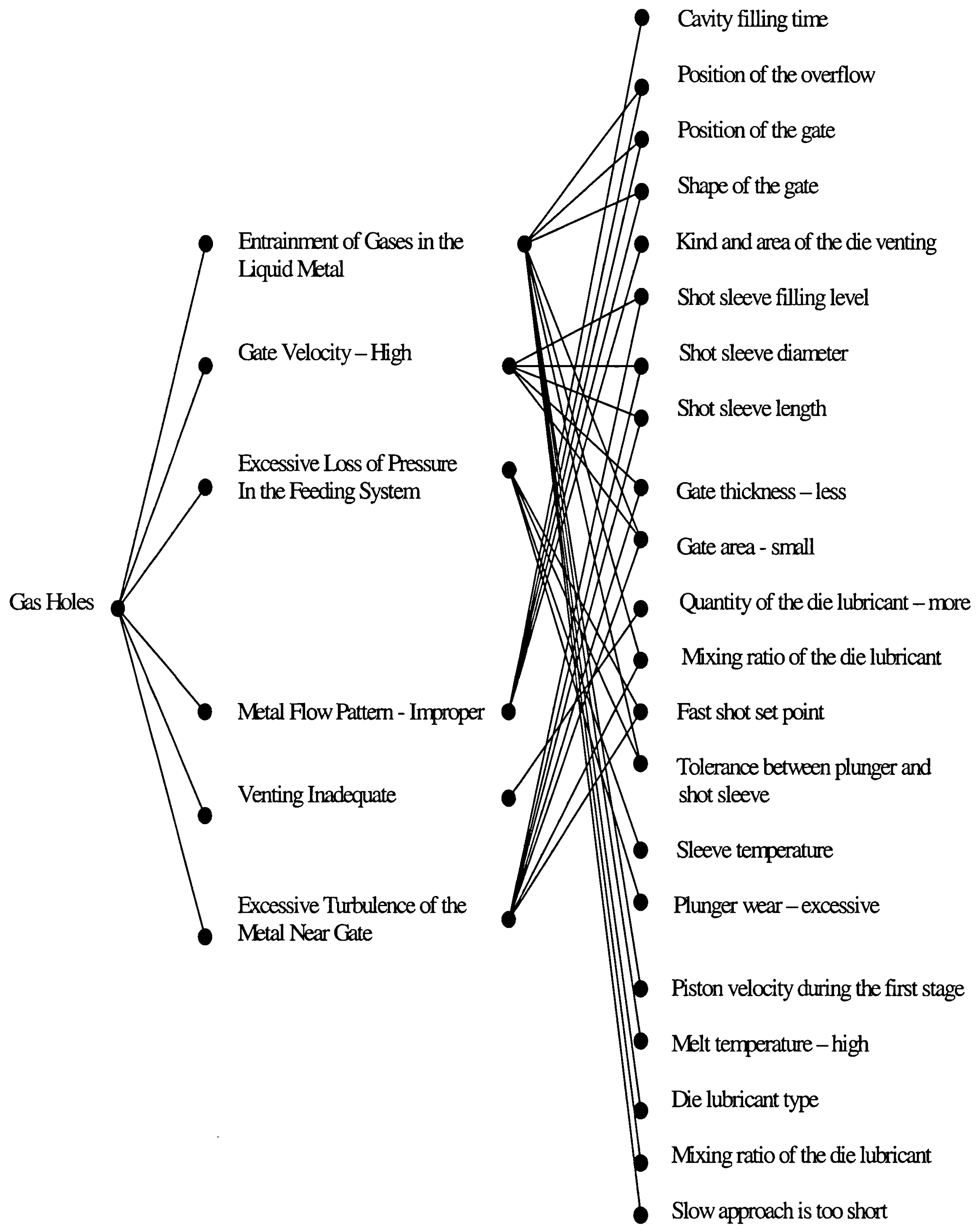

Gas Holes
Entrainment of Gases in the Liquid Metal
Gate Velocity – High
Excessive Loss of Pressure In the Feeding System
Metal Flow Pattern - Improper
Venting Inadequate
Excessive Turbulence of the Metal Near Gate
Cavity filling time
Position of the overflow
Position of the gate
Shape of the gate
Kind and area of the die venting
Shot sleeve filling level
Shot sleeve diameter
Shot sleeve length
Gate thickness – less
Gate area - small
Quantity of the die lubricant – more
Mixing ratio of the die lubricant
Fast shot set point
Tolerance between plunger and shot sleeve
Sleeve temperature
Plunger wear – excessive
Piston velocity during the first stage
Melt temperature – high
Die lubricant type
Mixing ratio of the die lubricant
Slow approach is too short

Figure 2.2: Defect-Metacause-Rootcause Representation for a defect "*Gas Holes*" for a Pressure Die Casting Process.

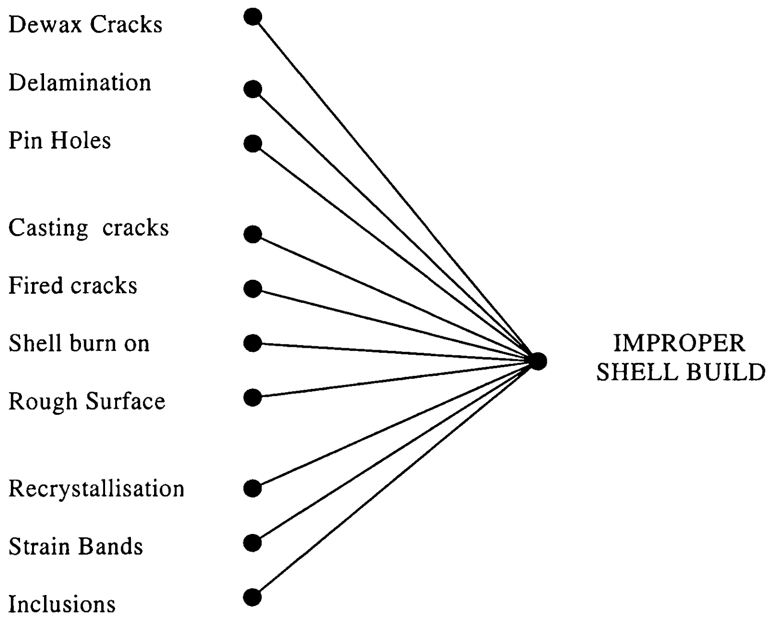


Figure 2.3: An Example of a List of Defects caused by "Improper Shell Build" for an Investment Casting Process.

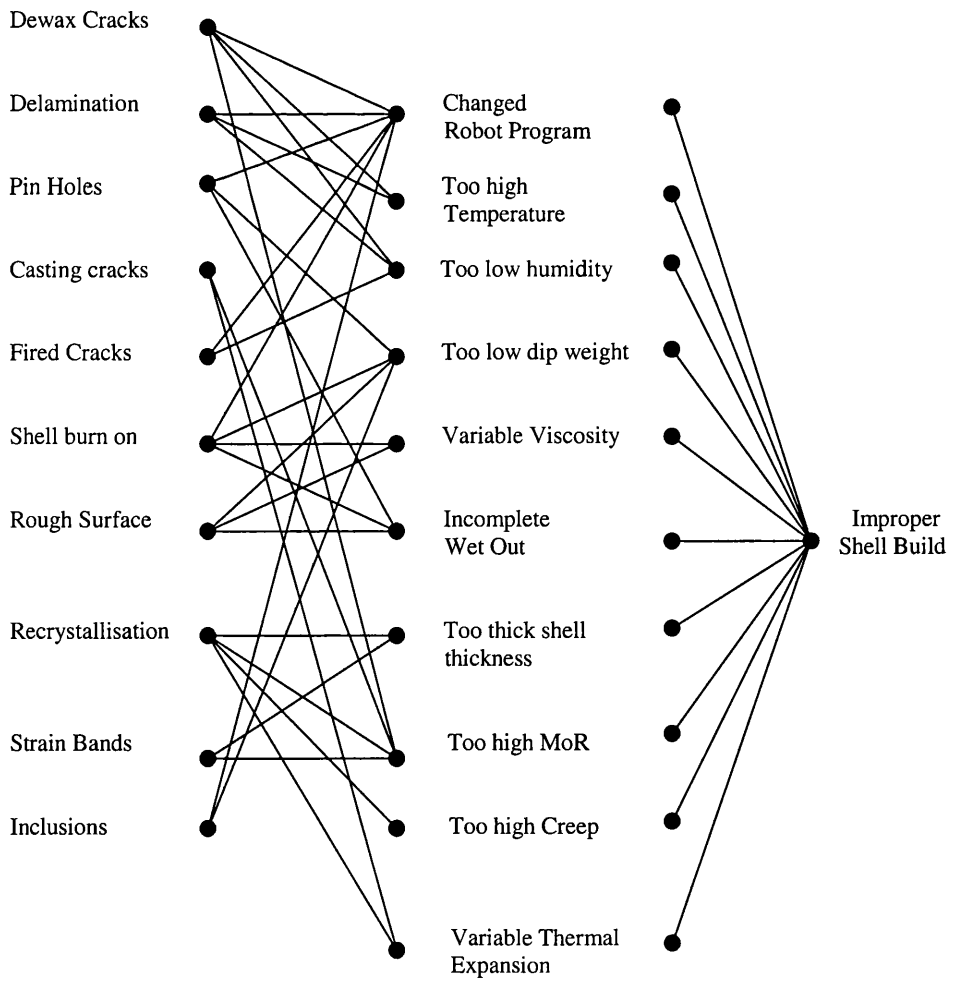


Figure 2.4: An Example of a List of Defects caused by "*Shell Build*" with metacauses for an Investment Casting Process.

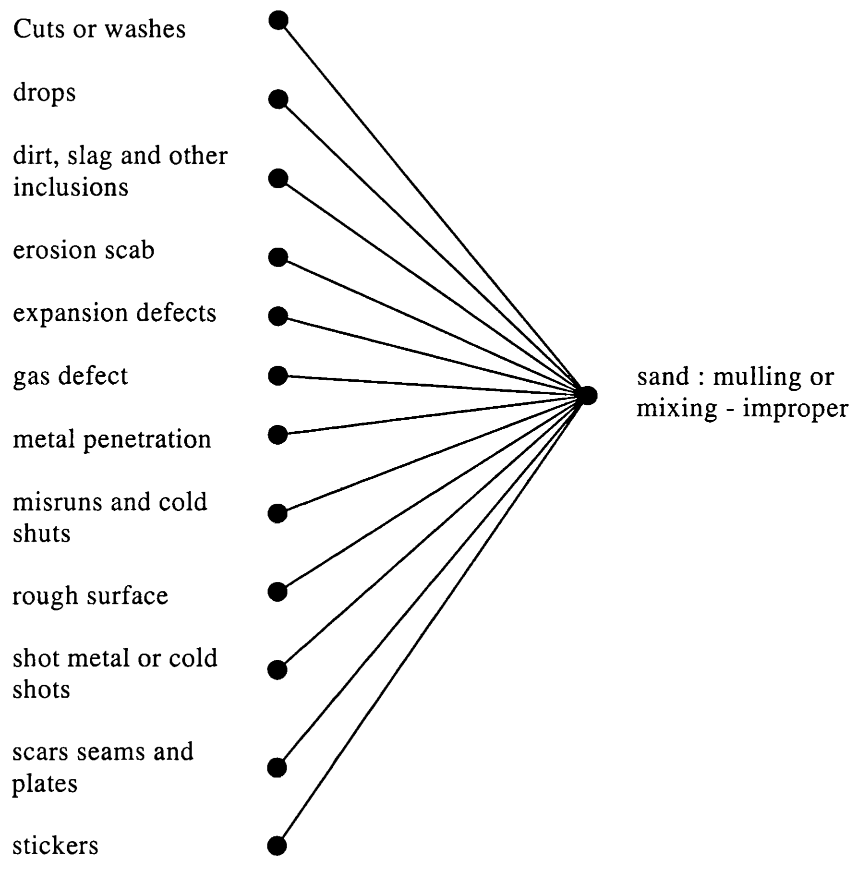


Figure 2.5: An Example of a List of Defects caused by the cause "*Sand: Mulling or Mixing – Improper*" for a Sand Casting Process.

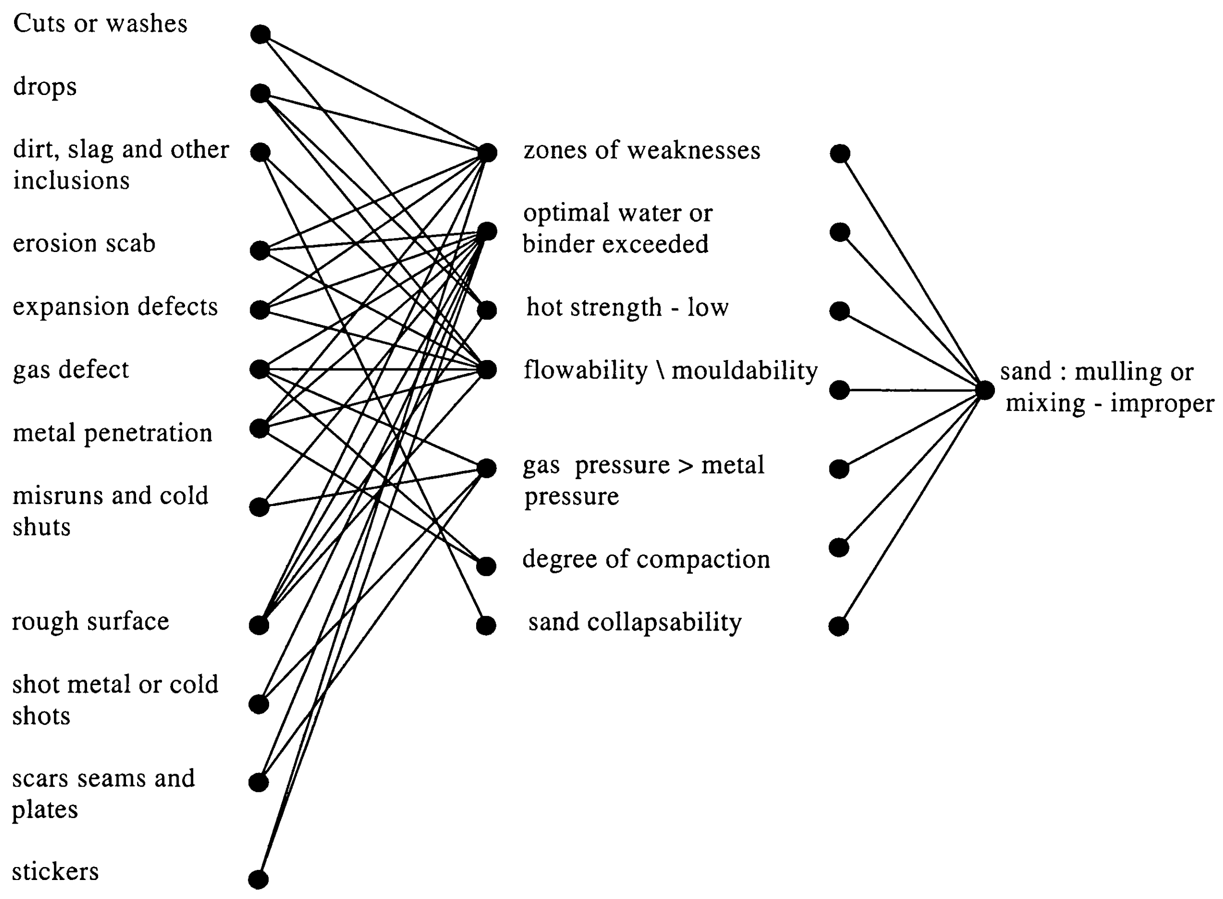


Figure 2.6: An Example of a List of Defects caused by the cause "*Sand: Mulling or Mixing – Improper*" with metacauses for a Sand Casting Process.

2.4 Input and Output of Network Models

As described in the second section of this chapter, the rationale behind the diagnosis is to identify a pattern in the rejection data (comprising the number of castings rejected under each defect type) and subsequently map the pattern onto a combination of causes, or a possible list of causes. As the cause and effect relationship is highly inter-linked, the occurrence and non-occurrence of defects determine the occurrence and non-occurrence of causes. Similarly, for a semantically constrained neural network, as the defect-metacause-rootcause relationship is highly inter-linked, the occurrence and non-occurrence of defects determine the occurrence and non-occurrence of metacauses, which in turn decide the occurrence and non-occurrence of rootcauses. The activations for all the nodes are scaled from 0 to 1. A value of zero corresponds to the non-occurrence of the nodes and a value of one corresponds to the occurrence of the node and the fractions denote the strength of occurrence of the corresponding node.

The rejection data consists of the number of defective components for each defect type. A component showing two defect types is counted under both defect types. The defect type with the maximum number of components is assigned a unit value and the rest of the defect types were proportioned accordingly to the scale zero to one. In this way, the relative strength of each defect is calculated. This forms an input vector to the network. The activations in the output nodes, each corresponding to a particular cause, would generate the diagnostic output for the network model.

The authors' objective for this research is to elicit the diagnostic process and particularly, understand the cause and effect relationships as follows:

1. To investigate whether any physical insight can be sought in the way neural networks would map rejection patterns to a combination of causes.

2. To study whether a neural network 'learning by examples' paradigm could be used to automatically quantify causal linkages.

3. To discover the function F as described in the second section of this chapter.

2.5 Conclusion

The "*cause and effect*" relationship in castings is highly complex and non-linear. The rejection data for a given casting and time frame, normally indicates a pattern, which has a few defects occurring at significantly high proportions and some occurring at significantly low proportions. The diagnostic problem, therefore, was defined as recognising the patterns in the rejection data and correspondingly associating them with a combination of causes.

Two approaches have been explored for undertaking the diagnosis of defective castings: The rule-based and the neural network approaches. These approaches have been discussed in this chapter so as to define the diagnostic problem and identify research goals for this thesis.

The rule-based expert system approach requires knowledge of the degree of influence of the related cause on the occurrence of each defect. In the authors' experience, generating such a probability distribution for the entire relationship is extremely difficult if not impossible. The second approach i.e. the network-based approach, offers a convenient computational tool, which, unlike the rule based approach, can adapt, learn from past examples and may then be used to quantify highly complex and inter-linked relationships. Therefore, a neural network approach has been explored for analysing the cause and effect relationships. The defect-metacause-rootcause type of causal representation has been illustrated with sample examples. Such a representation of causal knowledge has certain definitive advantages over a rule-based representation of causal relationships. Hence, the objective of this research is to study whether a neural network 'learning by examples' paradigm could be used to automatically quantify causal linkages.

REFERENCES

[1] Creese R. C., "Introduction to Expert Systems to Foundry Application", *AFS Trans,* **96**, 1988, pp 443-446.

[2] Er. A., Kondic V., "Knowledge-based systems and their application in casting defects control", *International Journal of Cast Metals Research*, vol **9**, No. **3**, 1996, 163-182.

[3] Piwonka T. S., Current and Potential Use of Process modeling for Foundry Process Control, *AFS trans*, Vol. **97**, 1989, pp 465-472.

[4] Phelps T. A., "Analysis of Internal Unsoundness Casting Defects using Artificial Intelligence Techniques", *AFS Trans*, Vol **97**, 1989, pp 507-512.

[5] Roshen H.Md., "Expert System for Analysis of Casting Defects: Cause Module", *AFS trans*, Vol. **97**, 1989, pp 601-606.

[6] Ransing R. S. and Shrinivasan M. N., "Computer Aided Analysis for the Identification of Causes of Defects in the Foundry", *40th Annual Convention of IIF*, Madras, India, 1992, pp 285-291.

[7] Ransing R. S., "Intelligent Computer Aided Defect Analysis for Casting", *Masters Thesis*, Dept. of Mech. Engg., Indian Institute of Science, Bangalore, India, 1992.

[8] Ransing R. S. and Lewis R. W., "Analysing the Quality of Die Cast Components using Expert Systems Related Technologies", *EDP Congress 1994, 123rd TMS Annual Meeting*, San Francisco, USA, 1994, pp 887-898.

[9] Ransing R. S., Shrinivasan M. N. and Lewis R. W., "ICADA:Intelligent Computer Aided Defect Analysis for Castings", *Journal of Intelligent Manufacturing*, vol **6**, no. **2**, 1995, 29-40.

[10] Ransing R. S. and Lewis R. W., Connectionism and Manufacturing Diagnosis, *Int. Conference on Advances in Mechanical Engineering*, Bangalore, India, 1995, Supplement 129-147.

[11] Ransing R. S., "An approach for the Causal Analysis in Casting Processes based on the Probabilistic Annalysis, Neural Network and Design Optimisation", *Ph D Thesis*, Inst. of Numerical Methods in Engg., University of Wales Swansea, Swansea, UK, 1996.

[12] Lewis R. W. and Ransing R. S., "Semantically Constrained Bayesian Network for Manufacturing Diagnosis", *International Journal of Production Research*, 1997.

[13] Ransing R. S. and Lewis R. W., "A Semantically Constrained Neural Network for Manufacturing Diagnosis", *International Journal of Production Research*, 1997.

[14] Askeland D. and Patel D., "Factorial Analysis of Some Variables and Their Optimisation for the Production of Sound Die Casing", Proc. 9th International Die-Casting Exposition and Congress, 1977, paper no. G-T77-013.

[15] Wightman D. E., Temperature Control in the Die-Casting Process Through the Use of a Heat Transfer System, Proc. 9th International Die-Casting Exposition and Congress, 1977, paper no. G-T77-066.

[16] Takach B. V., Some Aspects of Feed Design for Pressure Die-Casting Dies, Proc. 10th International Die-Casting Exposition and Congress, 1979, paper no. G-T79-094.

[17] Shortliffe E. H., "Computer based medical consultations: MYCIN", New York: American Elsevier, 1976.

[18] Davis R., Buchanan B. G. and Shortliffe E. H., "Production rules as representation for a knowledge-based consultation program", *Artificial Intelligence*, **8**, 1977, 15-45.

[19] Pandelidis I. O. and Kao J. F., "DETECTOR: A knowledge-based system for injection moulding diagnostics", *Journal of Intelligent Manufacturing*, **1**, 1990, 49-58.

[20] Zedha L. A., "Fuzzy Sets", *Information and Control,* **8**, 1965, 338-353.

[21] Smith A. E., "Predicting product quality with backpropagation: a thermoplastic injection moulding case study", *International Journal of Advanced Manufacturing Technology*, **8**, 1993, 252-257.

[22] Smith A. E. and Dagli C. H., "Controlling industrial processes through supervised, feedforward neural networks", *Computers and Industrial Engineering*, **21**, 1991, 247-251.

[23] Lin H., Yih Y. and Salvendy, "Neural network based fault diagnosis of hydraulic forging presses in China", *International Journal of Production Research*, vol **33**, no **7**, (1995) 1939-1951.

[24] Martinez E. E., Smith A. E. and Idanda B., "Reducing waste in casting with a predictive neural model", *Journal of Intelligent Manufacturing*, vol **5**, no. **4**, 1994, 277-286.

[25] Spelt P. F., Knee H. E., Glover C. W., "Hybrid artificial intelligence architecture for diagnosis and decision-making in manufacturing", *Journal of Intelligent Manufacturing*, vol. **2**, 1991, 261-268.

[26] Webster C. A. G., "The Use of Images in a Language-Independent and Culture-Independent Expert System for Diagnosing Pressure Die Casting Defects", *Journal of Materials Processing Technology*, vol **55**, no **3-4**, 1995, pp 296-302.

[27] Zhang H. C., Huang S. H., "Aplications of neural networks in manufacturing: a state-of-the-art survey", *International Journal of Production Research*, vol **33**, no **3**, 1995, pp 705-728.

Chapter 3

Feed Forward Neural Networks

In this chapter the basic structure and function of neural networks is introduced. The concept of neural networks is based on the human nervous system, which is built of cells called as neurons. This chapter introduces the feed-forward neural networks, which are the most commonly used networks, in which nodes (such as input, hidden and output nodes) are arranged in layers such that a node is only connected to nodes in the next layer. These networks are also classified as single and multi layer neural networks according to the number of hidden nodes in the network. A numerical value referred to as weight, is assigned to every connection between two nodes of consecutive layers. The process of learning or training a network is a methodology for changing weights within the network so as to achieve desired network performance. The back propagation algorithm - one of the most widely used algorithms for neural network training is described and the effect of network parameters on the performance of the network is studied in detail.

3.1 Introduction

Neural networks are systems composed of many simple-processing elements whose functions are determined primarily by their pattern of connectivity. These systems are capable of high-level functions, such as adaptation or learning, along with lower level functions such as data processing for different kinds of inputs. Neural networks have been inspired both by biological nervous systems and mathematical theories of learning, information processing and control [1]. Historically, much of the inspiration in the field of neural networks came from the desire to produce artificial systems capable of sophisticated, perhaps "intelligent", computations similar to those that the human brain routinely performs.

The structure and function of neural networks is based on our current understanding of the biological nervous system. Neural networks are built on a large number of simple, interconnected and adaptable processing units. Recently, neural network technology has gained more popularity because, these units store practical knowledge through learning from examples and, like biological systems, have the ability to take in hazy information from the outside world and process it without an explicit set of rules. This approach is in contrast to traditional expert system techniques, which analyze information according to a set of exact rules.

3.1.1 Biological prototype

The concept of neural networks is similar to the human nervous system, which is built of cells called neurons. A neuron is the fundamental building block of the nervous system. The human nervous system is extremely complex in terms of interconnections of neurons. It is estimated that 10^{11} neurons participate in perhaps 10^{15} interconnections over transmission paths that may range for a meter or more [1]. Each neuron shares many characteristics with other cells in a body, but has unique capabilities to receive, process, and transmit electrochemical signals over the neural pathways that comprise the brain's communication system.

Figure 3.1 shows the structure of a pair of typical biological neurons, which consists of three sections: cell body, dendrites, and axon. Dendrites are branches that extend from the cell body (or the neuron A) to other neurons (e.g. neuron B). A neuron receives its input signals through dendrites (or a link connecting two neurons) at a connection point called a synapse. On the receiving side of the synapse, when traveling along these dendrites, input signals are conducted to the cell body, where they are aggregated ("summed"). An input may either excite the cell, or inhibit it's firing. When the cumulative excitation in the cell body exceeds a threshold, the cell fires, sending a signal down the axon to other neurons. The neural network is a simplistic model of the complex human nervous system and has been described in the next section with an analogy of biological neurons.

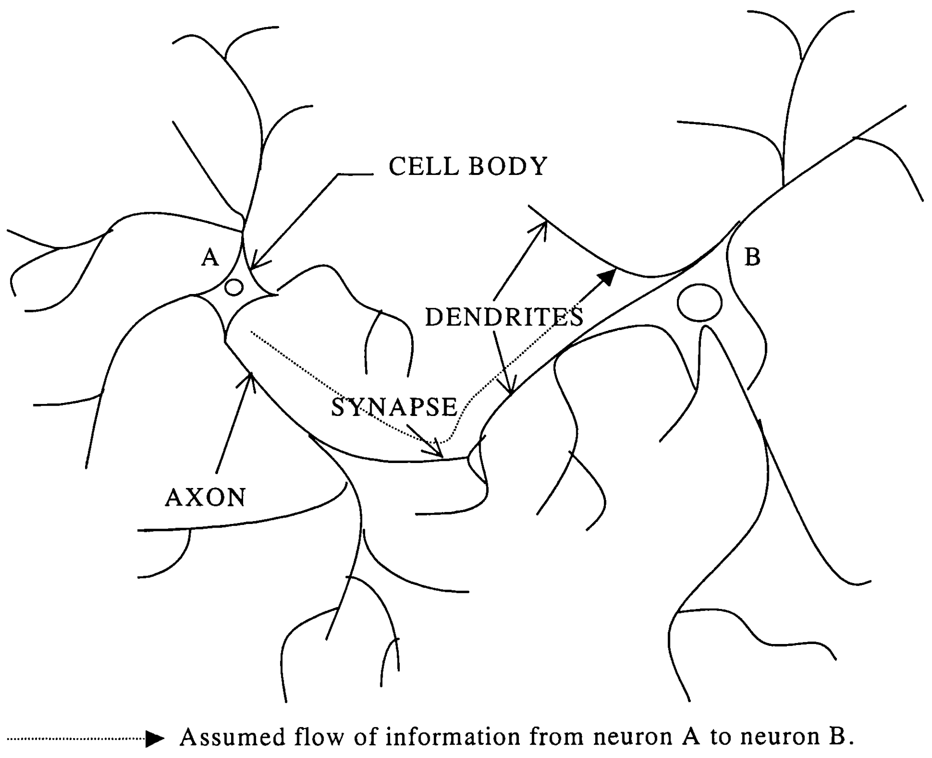


Figure 3.1 Neural Networks – The Biological Prototype

3.1.2 Basic Structure of an individual neuron in a neural network

An artificial neuron is designed to mimic the characteristics of a biological neuron. The block diagram of Figure 3.2 shows the model of a neuron, which forms a basis for designing (artificial) neural networks. Three basic elements of the model have been identified as follows:

1. As shown in Figure 3.2, $o_1,...o_i,..., o_l$ represent a set of input signals, which may also be an output of another neuron. The pre-synaptic activities are represented by these inputs corresponding to the signals into the synapses of a biological neuron. The figure also shows a set of synapses or connecting links, each of which is characterised by an adjustable parameter, called weight ($w_{1j},...w_{2j},...w_{ij},...w_{lj}$).

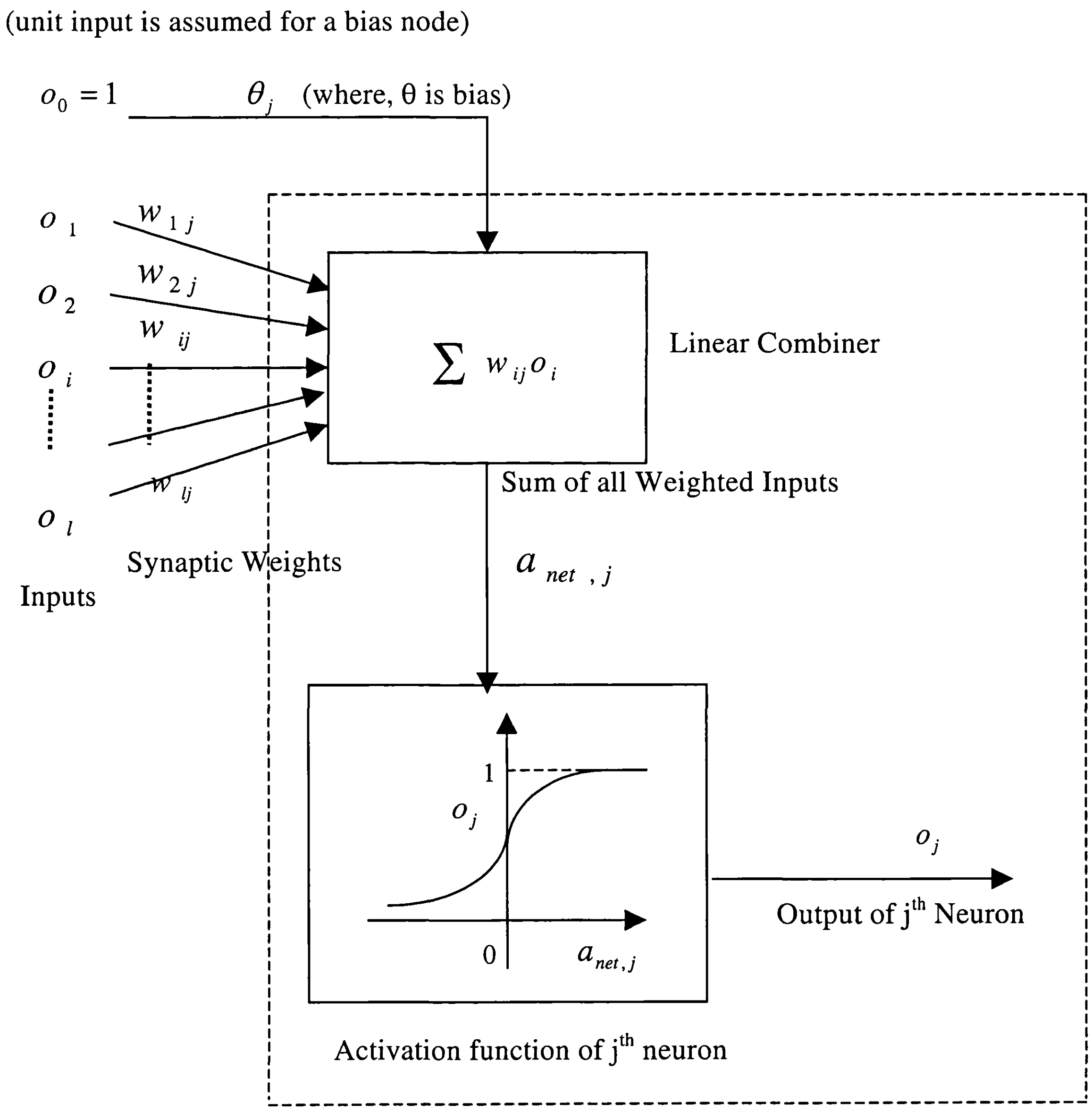


Figure 3.2 Model of a Neuron. Hashed rectangle encapsulates the processing involved within a neuron.

For an artificial neuron, a signal o_i at the input of synapse i connected to neuron j is multiplied by the corresponding synaptic weight w_{ij}, which is analogous to the synaptic strength. The first subscript refers to the input end of the synapse (connecting link) where as the second subscript refers to the neuron under consideration. Unlike a synapse in the brain, the synaptic weight of an artificial neuron may lie in the range that includes negative as well as positive values [2]. An external parameter referred to as bias θ_j, is applied to the unit j, which also contributes to the activation of the j^{th} node. The necessity and advantages of using the bias term are discussed in the following sub-section.

2. The Figure 3.2 also shows a linear combiner for summing the input signals, weighted by the respective synapses of the neuron. The summation of all weighted inputs is collectively called as the net input, $a_{net,j}$. This describes the total post-synaptic activity. A linear combiner in the artificial neuron replaces a dendrite in the human nervous system.

3. Net input $a_{net,j}$ is then passed through an activation function that is used for limiting the amplitude of the output of a neuron. The activation function is also referred to as a 'squashing function' because it squashes (limits) or normalizes the permissible amplitude range of the post-synaptic activity (output signal). Typically, the normalized amplitude range for an artificial neuron is either a closed unit interval [0,1] or alternatively [-1, +1]. Probably, the most commonly used activation function is a sigmoid activation function.
   Activation functions are also described later in this chapter. In general, for the j^{th} node, a sigmoid activation function is a function of the following variables, viz.

$$o_j = \frac{1}{1 + e^{-c_j a_{net,j}}} \qquad (3.1)$$

And the neural network model of Figure 3.2 is mathematically, expressed as:

$$a_{net,j} = \left(\sum_{i=1}^{l} w_{ij} o_i \right) + \theta_j \qquad (3.2)$$

where,

o_i	output signal from the i^{th} unit.
w_{ij}	weight of the link from unit i to unit j.
o_j	output of the j^{th} unit.
$a_{net,j}$	net input activation for the j^{th} unit.
θ_j	bias for the j^{th} unit.
c_j	gain describing the slope of the activation function.

3.1.3 Arrangement of Neurons in a network along with its associated parameters

The network parameters, which may influence the diagnostic abilities of a neural network, are discussed in detail in this section.

1. **Neurons (nodes):** As introduced in the previous section, a neuron is an elementary processing unit of neural networks, which performs most of the processing. The neurons in a network are generally classified into three categories: input, output and hidden nodes. The input and output nodes interface with the outside world where as hidden nodes, as the name indicates, are internal nodes within the network.

   i.**Input nodes**: Input to these nodes form an input to the network. The data for a given problem is generally filtered or processed before presenting to a neural network as input. Filtering process is problem specific, which can vary from simple 'normalisation' to a complex mathematical procedure (statistical). Input nodes are generally arranged in a layer referred to as an input layer (Figure 3.3).

ii. **Output nodes**: Nodes whose outputs represent the output of a network are called as 'output nodes'. All output nodes are also grouped together to form an output layer as shown in Figures 3.3 and 3.4.

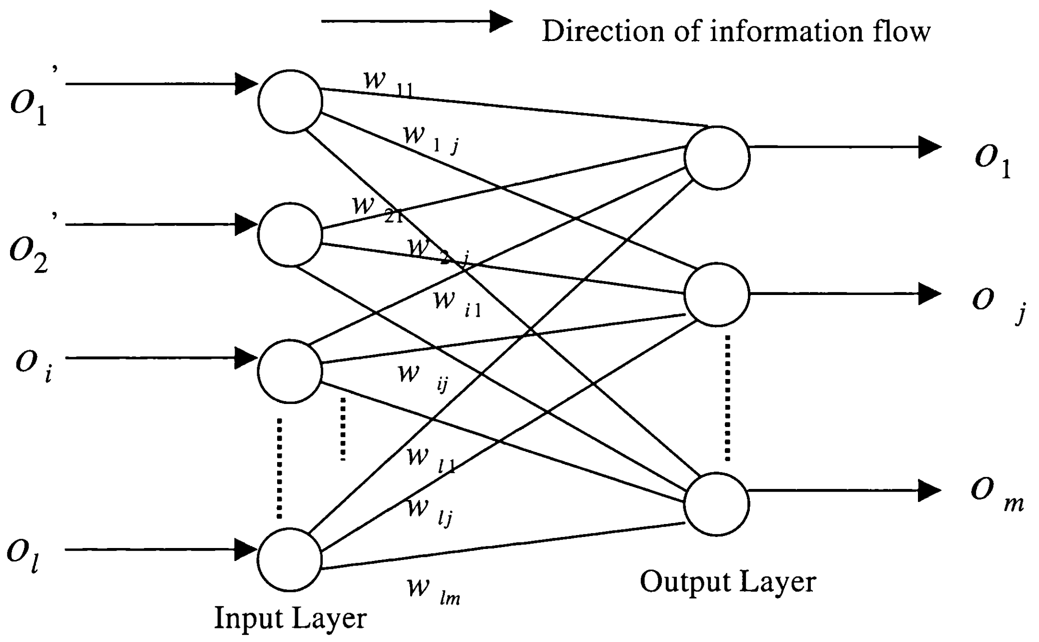


Figure 3.3 A Single-Layer Feed-Forward Neural Network

iii. **Hidden nodes**: The remaining nodes are called as hidden nodes because they are not visible from outside. These nodes receive inputs from other nodes in the network and their output form an input to the nodes in the next layer of the network. The hidden nodes are also organised in layers as shown in Figure 3.4.

2. **Weights**: In feed-forward neural networks, which are also the most commonly used networks, nodes are arranged in layers such that a node is only connected to nodes in the next layer. A numerical value is assigned to every connection between two nodes of consecutive layers, which is referred to as weight (Figure 1.3). The knowledge, which the neural network learns, is stored in terms of weights and is one of the most important variables in a network. A weight associated with two nodes indicates the strength of the association. The effect of the output of a node on the successor node is influenced by the value of the weight associated with the link

connecting the two nodes. In other words it means that the value of w_{ij} determines how strongly the output of the node *i* influences the activity of the node *j*. A weight can be exhibitory, inhibitory or neutral depending upon whether it is positive,

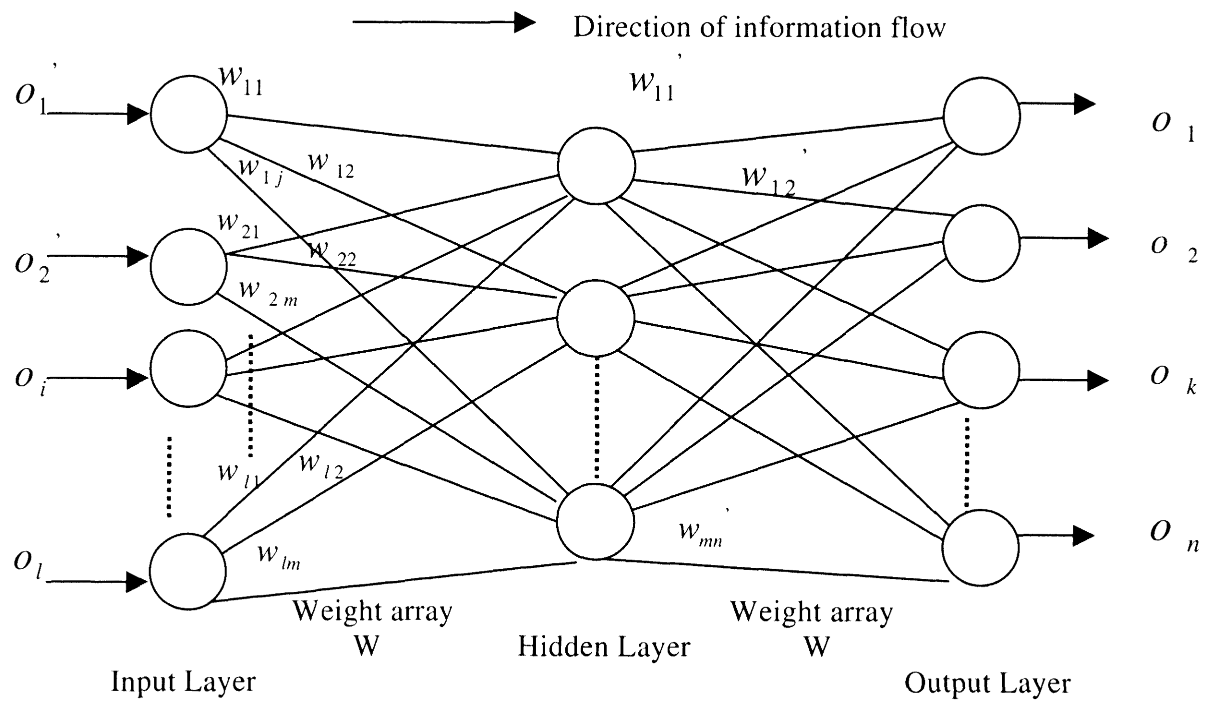


Figure 3.4 A Multi-layer Feed-forward Neural Network

$w_{ij} > 0$, negative, $w_{ij} < 0$, or zero, $w_{ij} = 0$ respectively. An inhibitory weight results in the reduction of an activation value at the target node (*j*) whereas an excitory weight increases the activation value. i.e. effectively enhancing the activity. In other words, the weight is a measure of how frequent a target node (*j*) has been active simultaneously with an input node (*i*).

During a training phase, a node is made adaptive to new information presented. This is also referred to as the learning process. In other words, the process of learning or training a network is nothing but a methodology for changing weights within the network so as to achieve desired network performance.

3. **Bias**: Many times it is also necessary to add an additional parameter referred to as bias (θ) to the weighted sum (Equation 3.2). As shown in Figure 3.5, the bias changes the value of the weighted sum at which the

sigmoid function changes it's output value from zero to one. In other words, the bias term does not restrict the decision hyper surface, which classifies the input space, to pass through the origin of the input co-ordinate system. This enhances the learning ability of a network. For the convenience of implementation, the effect of bias is incorporated by modifying the network as follows:

1. Adding a new input node with a fixed unit input value and
2. Adding a new synaptic weight equal to the bias θ.

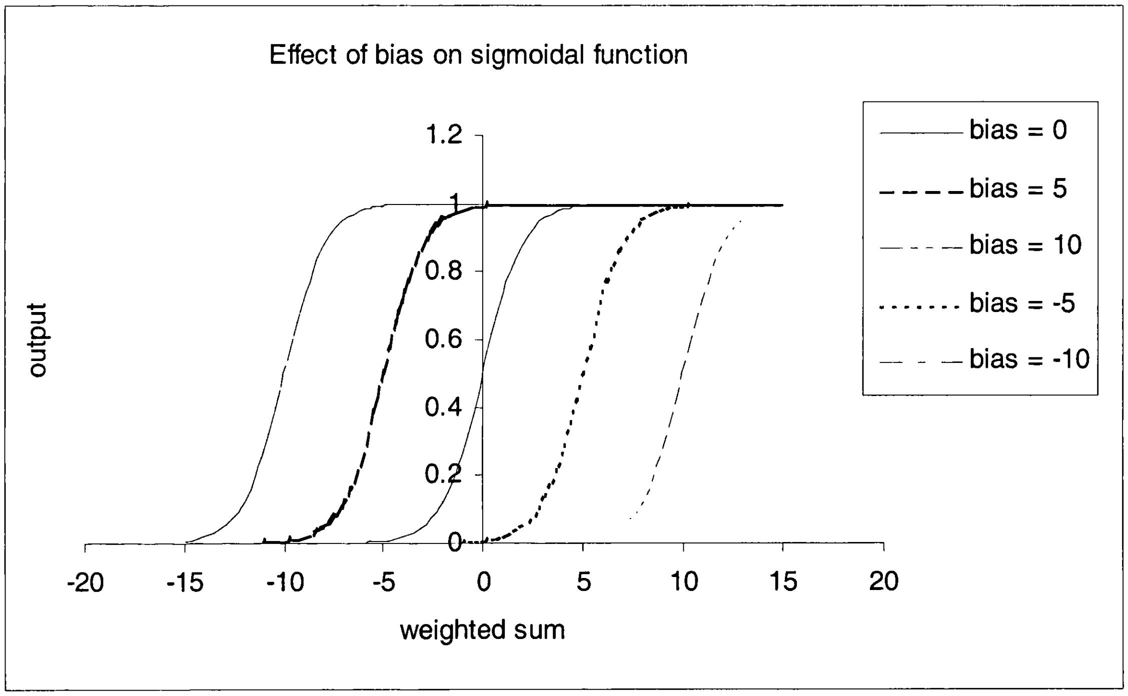


Figure 3.5 Effect of bias on the sigmoid activation function

3.2 Influence of the layered arrangement of neurons onto network's classification ability

3.2.1 Single-Layer Feed-forward Neural Networks

In the simplest form of a layered network, a layer of input nodes is connected to a layer of output nodes. Such a network (Figure 3.3) is called a single-layer network, where single-layer refers to the output layer of neurons. The input nodes do not perform any computation and are included in the diagram only for clarity. Therefore, it will not be counted as a layer in this work.

While describing a network, a set of inputs, labeled $o_1',...o_i,...,o_l$, is applied to the neural network and is collectively referred to as the input vector O. For a fully connected network, every node is connected to every other node in the next layer. Therefore, as shown in Figure 3.3, every input node is connected to all output nodes and the computation at output nodes is performed as shown in Figure 3.2.

3.2.2 Separability of Input Patterns and the Single Layered Networks

The ability of a single layered neural network to classify an input space has been demonstrated in this section with the help of a simple example (Figure 3.6). The network has two input nodes with inputs: x_1 *and* x_2, $x_1 \in [0,1]$ and $x_2 \in [0,1]$, respectively and one output node with output $o_j \in [0,1]$. For a linearly separable input pattern, the decision boundary, which partitions or classifies the input space into two regions, is linear as shown in Figure 3.7 and the equation of this line (or hyper line in a multidimensional input space) can be written as follows:

$$w_1x_1 + w_2x_2 + w_3 = 0 \qquad (3.3)$$

where, w_1 and w_2 are weights for the corresponding linkages to the output node and w_3 is an externally applied bias (Figure 3.6).

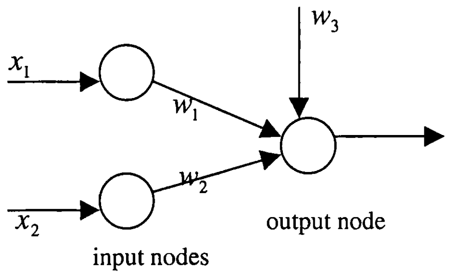


Figure 3.6 Single Layer Network

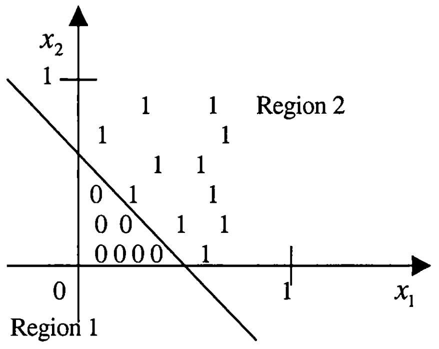


Figure 3.7 Linearly Separable Input Pattern

This network will associate a point (x_1,x_2) to 'region 1' (as shown in Figure 3.7) if the net input $w_1x_1 + w_2x_2 + w_3 < 0$ and to 'region 2' if the net input $w_1x_1 + w_2x_2 + w_3 > 0$. Note also that the effect of bias w_3 is merely to shift the decision boundary away from the origin as the slope of this line is determined by weights w_1 and w_2. Binary output values have been considered only for an illustration purpose so that the focus is retained on the linear-separability issue. If the single layer network correctly classifies a set of input signals into one of two regions, 1 or 2, the network is said to have learnt the training data. In other words, in neural network parlance, learning can be construed as an iterative procedure to arrive at an optimal decision boundary (or a hyper surface), which either correctly classifies the input space or maps every point in the input space correctly onto an output space. The mapping or the classification is done in such a way that the underlying relationship between input and output is learnt rather than merely memorising the training examples.

The linear decision boundary limits the use of single layer networks to classification problems in which the set of input and output points can be linearly separated. For a two-input case, as shown in Figure 3.6, the decision boundary is a straight line. For three input nodes, a plane will separate the resulting three-dimensional input space. For four or more inputs, although the visualization may break down, the n-dimensional input space is separated by an n-dimensional hyper plane.

However, if the two regions are as shown in Figures 3.8 and 3.9, the decision boundary becomes non-linear and the classification is beyond the computing capability of a single layer network. A multi-layered network as discussed in the next section then becomes necessary.

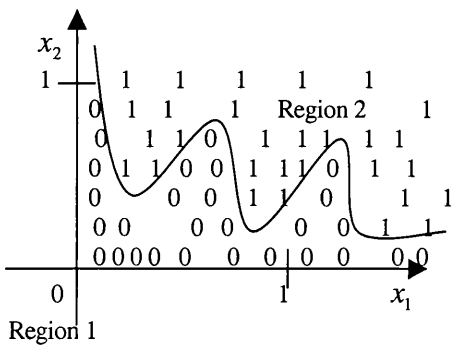


Figure 3.8

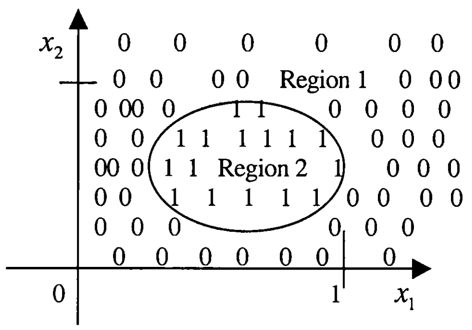


Figure 3.9

Nonlinearly Separable Input Patterns

3.2.3 Multi-layer Feed-forward Networks

Although the linear separability problem has been well understood since early stages of neural network development and it is also known that this serious representational limitation of single-layer networks could be overcome by adding one or more layers. This topic has been introduced in this chapter as it is used for analyzing cause and effect relationships using neural networks during this research.

The term multi-layer perceptron has often been used as a synonym for the term 'multi-layer feed-forward neural network', which represents a generalisation of a single layer perceptron. As shown in Figure 3.4, multi-layered neural networks are built by serially connecting layers of neurons. This class of feed-forward neural networks distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. The function of the hidden neurons is to intervene between external inputs, which are in fact the network input, and the network output in some useful manner, which has been discussed in detail in Chapter 5. In feed-forward networks, the input signal propagates from left to right throughout the network in a forward direction on a layer-by-layer basis.

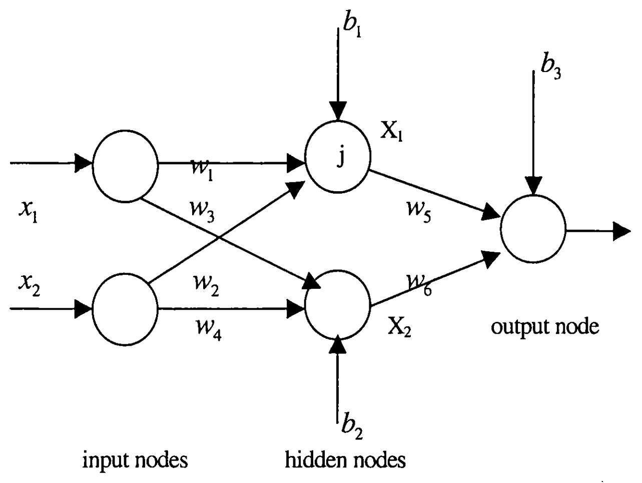
The output of a hidden unit is calculated in the same way as shown in Figure 3.2 which is based on the weighted sum of network input values. The output signal of the hidden layer is then used as an input to nodes in the next layer, which in this case is the output layer of the network. Typically, neurons in each layer of the network receive output values of nodes in the preceding layer as input values. The set of output values of neurons in the output layer of the network constitutes the overall response of the network to a given input pattern. In neural networks with more than one hidden layer, the first hidden layer is fed from input layer nodes. The resulting outputs from the first hidden layer are in turn applied as inputs to the second hidden layer and so on for the rest of the network.

The neural network in Figure 3.4 is a fully connected multi layer feed forward network in a sense that every node in each layer of the network is connected to every other node in the adjacent forward layer.

3.2.4 Separability of Input Patterns and Multi-Layered Networks

Consider a multi-layer network as shown in Figure 3.10 having two input nodes with inputs $x_1 \in [0,1]$ and $x_2 \in [0,1]$ respectively, one hidden layer with two hidden nodes and one output node with output $a_j \in [0,1]$. This network also partitions the input space into two regions, but the decision boundary can be a curve similar to the one shown in Figure 3.11. Other examples of non-linearly separable patterns have already been shown in Figures 3.8 and 3.9. Detailed discussion on the development of decision hyper surface using neural network algorithms has been undertaken in Chapter 5.

Such networks can also generate decision boundaries, which surround a single convex region of the input space. Networks having three layers of weights can generate arbitrary decision regions, which may be non-convex and disjoint.



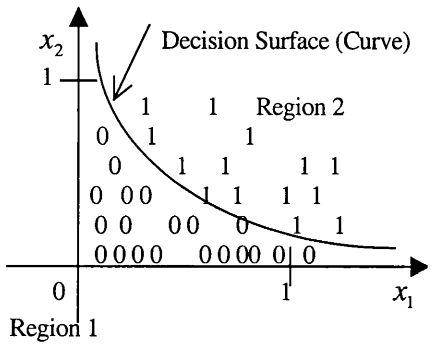


Figure 3.10 Multi-layer neural network (x_1, x_2 are inputs to input nodes and X_1, X_2 are outputs of hidden nodes)

Figure 3.11 Non-Linearly Separable Input Pattern

3.3 Activation functions

As shown earlier (Figure 3.2), the activation function generates an output for a node in a predefined range of (0,1) or (-1,1) as a function of the weighted inputs to that node. A number of activation functions have been used in the neural network algorithm. Among the popular activation functions are linear and sigmoid (logistic and 'tanh'). Typically, an activation function generates either unipolar or bipolar signals. A brief introduction to activation functions has been given in this chapter, as during this work time was also spent on analysing the shape of the sigmoid activation function. The discussion of activation functions in this section will help to maintain the continuity in later chapters.

1. **A linear function:** The output in the j^{th} node is same as the net input (weighted sum of input values plus the bias term) to this node.

$$o_j = a_{net,j} \qquad (3.4)$$

Such linear processing elements are studied in the theory of linear systems, for example in the "traditional" signal processing and the statistical regression analysis. They are also commonly used for output layer nodes, because the desired output is known during the training phase and it is not necessary to explicitly normalize the output.

2. **Step function:** The step function is the simplest and the most straightforward.

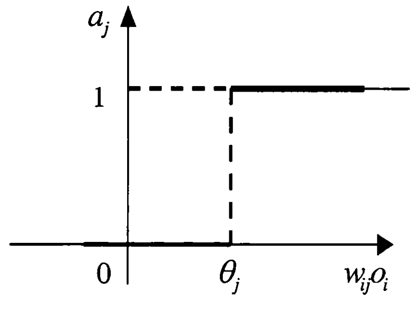


Figure 3.12 Unipolar Step Function

Neural networks with threshold units were studied by Rosenblatt (1962) under the name *perceptrons* where it was used as a threshold element with a binary output. Widrow and Hoff (1960) also called these units as *addalines.*

The bias θ_j (threshold) can be added to both unipolar and bipolar step functions depending upon which unit produces an output signal of either '1' or '0'. We then say that a neuron is "fired", when the synaptic activity exceeds the threshold level, θ_j for unit j. The step function has only binary output. Output of such a neuron is defined as follows:

$$a_j = \begin{cases} 1 & if\ a_{net,j} \geq 0 \\ 0 & otherwise \end{cases} \tag{3.5}$$

where, $$a_{net,j} = \sum_{i=1}^{l} w_{ij} o_i + \theta_j$$

Such a neuron, in the literature, is referred to as the *McCulloch-Pitts model,* in recognition of the pioneering work done by McCulloch and Pitts (1943) (Figure 3.12). In this model, the output of a neuron takes on the value of 1, if the net input i.e. $a_{net,j}$ of that neuron is nonnegative, and 0 otherwise.

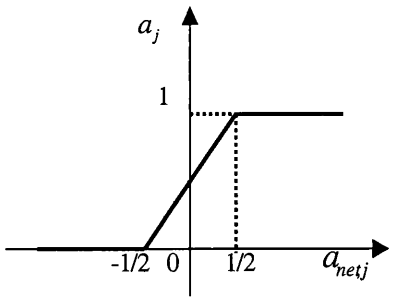
This statement also describes the *all-or-none-property* of the McCulloch-Pitts model.

3. **Ramp function (Piecewise-Linear function):**

In this type of activation function (Figure 3.13), the threshold or a ramping function mirrors the input within a given range, say 0 to 1, and functions as a hard limiter (step function) outside that range. It is a linear activation function that has been clipped to the minimum and maximum values, which then makes it non-linear. It gives an 'on' or 'off' response for activations well above or below the threshold value and will still give useful information if the activation is close to the threshold value. The function is defined as follows:

$$a_j = \begin{cases} 1 & \text{if} \quad a_{net,j} \geq +\frac{1}{2} \\ a_{net,j} & \text{if} \quad +\frac{1}{2} > a_{net,j} > -\frac{1}{2} \\ 0 & \text{if} \quad a_{net,j} \leq -\frac{1}{2} \end{cases} \tag{3.6}$$

where, $a_{net,j} = \sum_{i=1}^{l} w_{ij} o_i + \theta_j$



3.13 Ramp Function

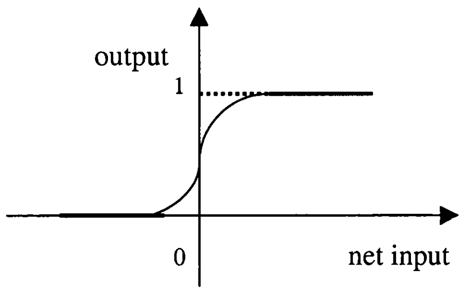


Figure 3.14 Logistic Sigmoid

The following three situations may be viewed as special forms of the piecewise-linear function.

1. For small activation potential, the neuron works as a linear combiner.
2. For large activation potential, the neuron saturates and generates the output signal of either 0 or 1.

3. For large gains $c \to \infty$, the piecewise-linear function is reduced to a step function.

The main disadvantage of this function is that the derivatives are discontinuous. A smooth S-shaped sigmoid function (Figure 3.14) overcomes this limitation and hence is widely used in neural network algorithms.

4. **Sigmoidal Functions:** The term sigmoid means 'S-shaped', and the logistic form of the sigmoid maps the interval $(-\infty,\infty)$ onto (0,1) for unipolar functions and (-1,1) for bipolar functions.

(a) Unipolar: The logistic sigmoid function is an unipolar function which is defined as

$$a_j = \frac{1}{1+e^{-c_j a_{net,j}}}$$

where, $$a_{net,j} = \sum_{i=1}^{l} w_{ij} o_i + \theta_j$$

and c_j is a constant value referred to as 'gain' which determines the slope of the sigmoid function.

This function is best suited for neural networks because it has advantages of the ramp and/or step activation functions and also has the following useful properties.

- It is a smooth and continuous function (this property is useful in deriving the weight updation expressions of the traditional neural networks).
- The function outputs values within the range 0 to 1. Traditional neural network generally has binary output of either 0 or 1. However, by using a sigmoid activation that outputs within the range 0 to 1, neural networks can also have fractional output values.

- The rate of change of output values for a given change of input $a_{net,j}$ is minimum at output values close to 0 and 1 and maximum at output values close to 0.5. Such exponential variation also assists the learning process.
- If $a_{net,j}$ is close to 0.5, then the logistic sigmoid function can be approximated by a linear function, and so in this sense a network with sigmoidal activation functions contains a linear network as a special case.
- The parameter gain c_j describes the slope of the activation function. By varying c_j, sigmoid functions of different slopes are obtained. As the slope parameter gain, c_j gets larger, the sigmoid approaches the behavior of the step function. The effect of gain on the sigmoid function can be seen in Figure 3.15.

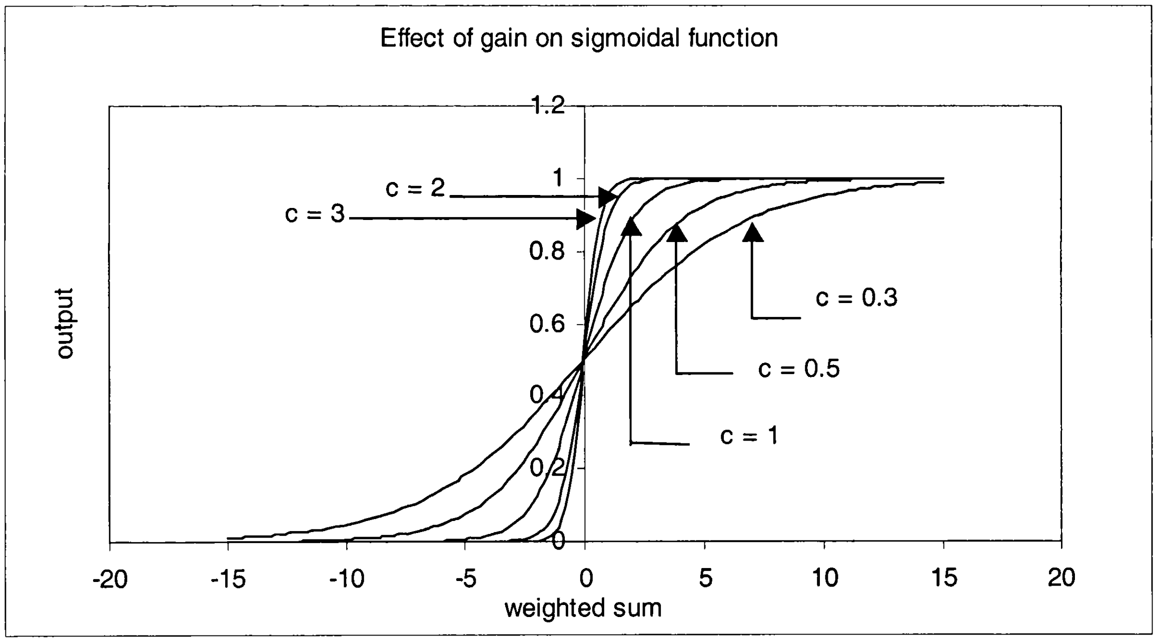


Figure 3.15 Effect of gain on the sigmoid activation function

<u>(b) Bipolar:</u> The activation functions defined in Equations 3.1, 3.5 and 3.6 range from 0 to +1. Another popular function has output in the range –1 to +1 as shown in Figure 3.16. The shape is anti-symmetric with respect to origin; i.e. the activation function is an odd function of the weighted sum. This bipolar sigmoid activation function is also known as the Anti-symmetric Sigmoid or the Hyperbolic Tangent function, mathematically defined as

$$a_j = \tanh\left(a_{net,j}\right) = \frac{e^{ca_{net,j}} - e^{-ca_{net,j}}}{e^{ca_{net,j}} + e^{-ca_{net,j}}} \qquad (3.7)$$

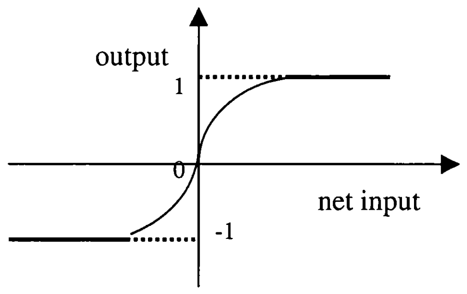


Figure 3.16 Hyperbolic Tangent

The hyperbolic tangent (bipolar sigmoidal) function has also been widely used, specifically, in problems related to function mapping and approximation. It is often found that 'tanh' activation functions give rise to faster convergence of training algorithms than logistic functions, which has been further discussed in Chapter 4.

3.4 Neural Network Learning

The neural network learning process is accomplished in two stages.

1. **Network Training**: During this stage, the network parameters such as weights, biases, number of hidden nodes and hidden layers are determined by presenting an example set of known input and output pairs and minimizing the error between network prediction and known output during each presentation. The optimal values of network parameters are then used for testing the network.

2. **Network Testing**: During the training stage, care needs to be taken such that the network does not merely memorise the training data but instead learns the underlying relationship between input and output values. The performance of the network is tested on another set of known input and output examples. The values in testing data set are different from the training data set and are used to assess generalization abilities of the chosen

neural network architecture. If the network performance is poor on the testing data, the network is retrained on the training data.

During training stage, the network parameters are updated either with supervised, or unsupervised learning algorithms as described in the following sub-section.

3.4.1 Supervised Learning

In a supervised learning process, one may think of a supervisor (teacher) as having knowledge of the environment, with that knowledge being presented by a set of input-output examples. The environment is, however, unknown to the neural network. The supervisor and the neural network are both exposed to an input vector. By virtue of the built-in knowledge, the supervisor is able to provide the neural network with a desired response for that input vector. Indeed, the desired response represents the optimum action to be performed by the neural network. The training process (Figure 3.17) is explained as follows.
The network parameters are adjusted under the combined influence of the input vector and the error signal. The *error signal* is defined as the difference between the desired response and the actual response of the network. The adjustment is carried out iteratively in a step-by-step fashion according to an algorithm that tends to minimize this error. This procedure is carried out until the error for the entire training set attains an acceptably low level, which may not necessarily mean a minimum value. This completes the learning procedure and the weights are then "frozen" and the network performance is tested on the testing data before its actual use.

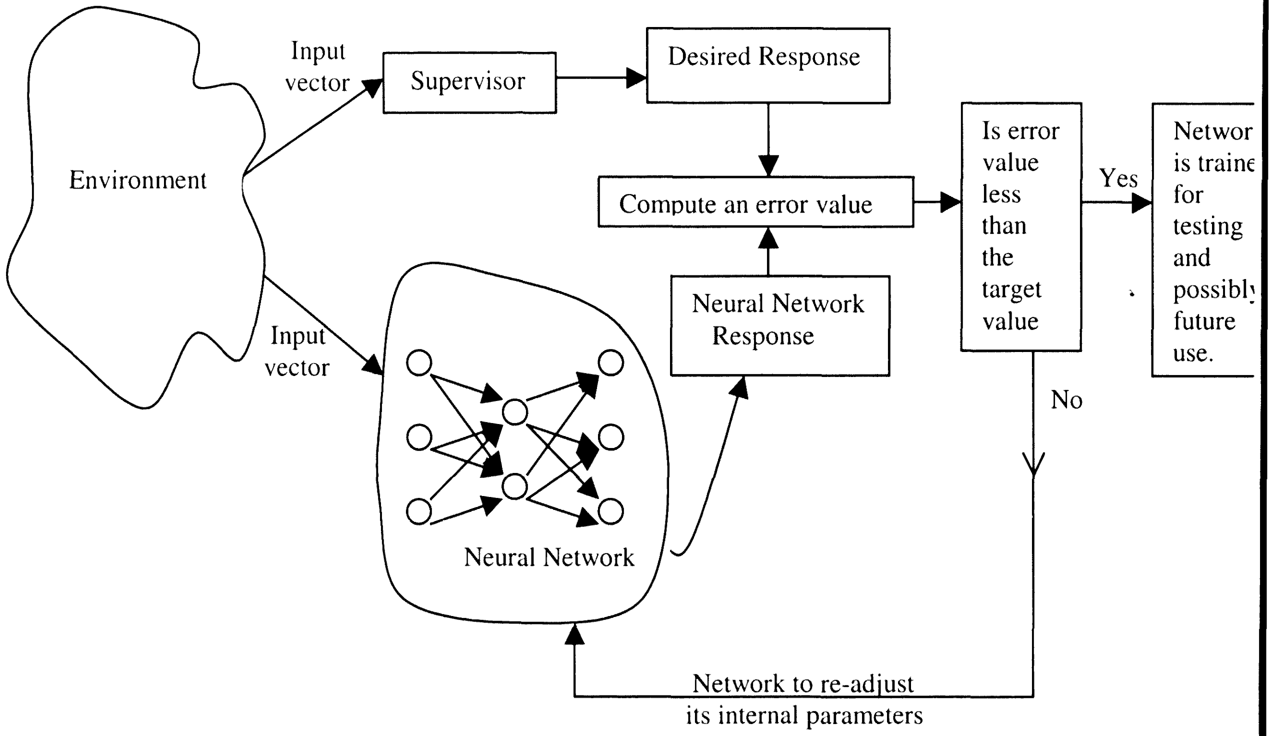

Environment
Input vector
Supervisor
Desired Response
Compute an error value
Is error value less than the target value
Yes
Networ is traine for testing and possibl future use.
No
Input vector
Neural Network Response
Neural Network
Network to re-adjust its internal parameters

Figure 3.17 Supervised Learning

The aim is to eventually make the neural network *emulate* the teacher. In this way, the knowledge of the environment available to the teacher is transferred to the neural network through training so that it is able to deal with the environment completely by itself. A supervised learning system is usually able to perform tasks such as pattern classification and function approximation. Back-propagation learning method – most widely used example of supervised learning – has been described in Section 3.5.

3.4.2 Unsupervised Learning (e.g. competitive learning)

Unsupervised learning is sometimes referred to as self-supervised learning. As the name implies, there is no external supervisor or teacher or critic to oversee the learning process as indicated in Figure 3.18. Rather, the learning process is accomplished based on local information only. As target output is not known, the training set consists only of input vectors. The network looks for regularities or trends in input signals, and makes adaptations according to the function of the network. In this process, the network extracts underlying correlations within input examples and groups similar vectors and creates new classes automatically (Becker, 1991). This process also needs an internal monitoring of performance.

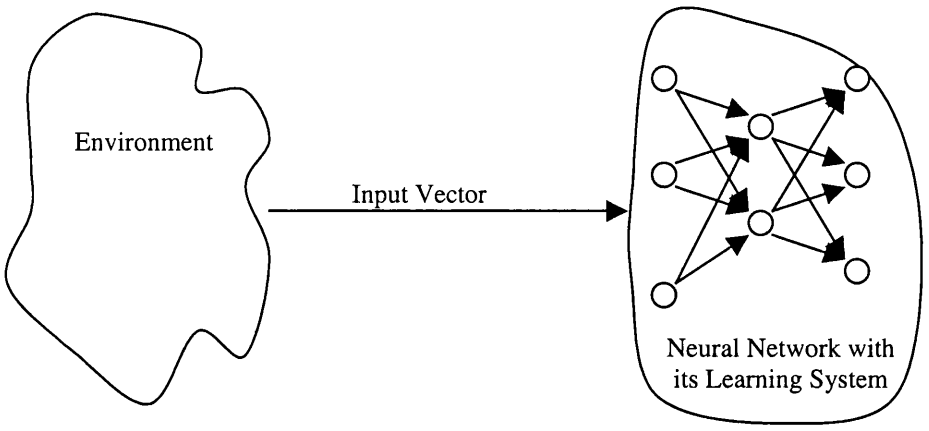


Figure 3.18 Unsupervised Learning

Competitive learning rule is an example of the unsupervised learning process. For example, we may use a neural network with an input layer and a competitive layer. The input layer receives the available data. The output or the competitive layer consists of neurons that compete with each other (in accordance with the competitive learning rule) for an opportunity to respond to the features contained in the input data. The neuron with the greatest total input wins the competition and is called a "*winner-takes-all*" neuron. All other neurons then switch off. The output signal of winning neuron is set equal to one and the output signals of all the neurons that lose the competition are set equal to zero. An important feature of competitive learning is that only a single output neuron is active at any one time. It is this feature that makes competitive learning highly suited to discover statistically salient features that may be used to classify a set of input patterns.

3.5 Back-Propagation Algorithm (Supervised Learning)

Most training algorithms involve an iterative procedure for the minimization of an error function, with adjustments to weights being made in a sequence of steps. The procedure for supervised error-back-propagation is as follows:

1. Start the cycle by presenting input patterns to the neural network.
2. Specify desired outputs for each input pattern.
3. This input is then propagated forward through the network, layer-by-layer, as per Equations 3.1 and 3.2 until the output layer.
4. A set of output produced is considered as the actual response of the network. Steps 1, 2, 3 and 4 constitute the "**Forward Propagation Phase**" in that the signal propagates from nodes in the input layer to nodes in the output layer.
5. Error is calculated by comparing network output with the desired output as follows.

$$E = \frac{1}{2}\sum_{k=1}^{n}(t_k - o_k)^2 \qquad (3.8)$$

where,

n:	number of output nodes in the output layer.
t_k:	Desired output of the k^{th} output unit.
o_k:	Network output of the k^{th} output unit.

6. The error signal (E) is propagated backwards through the network and is used to adjust weights. The weights in the links connecting to output nodes (w_{jk}) are then modified based on the gradient descent method as follows.

$$\Delta w_{jk} = \eta\,(-\frac{\partial E}{\partial w_{jk}}) \qquad (3.9)$$

$$= \eta \delta_k o_j$$

where,

o_j	Output of the j^{th} hidden unit.

The error is propagated backwards to compute the error specifically, at the hidden nodes.

$$\Delta w_{ij} = \eta \left(-\frac{\partial E}{\partial w_{ij}}\right) \quad (3.10)$$

$$= \eta \, \delta_j o_i$$

where,

o_i	Output of the i^{th} input unit (which is same as the output value).
η	learning rate value
i, j, k	Subscripts i, j and k correspond to input, hidden and output nodes respectively
w_{jk}	Weight on the link from unit j to k.
w_{ij}	Weight on the link from unit i to j.
δ_k	$= o_k(1-o_k)(t_k - o_k)$... for output units.
δ_j	$= o_j(1-o_j)\sum_k \delta_k w_{jk}$ ... for hidden units.

In this way, the error is propagated backwards to modify weights so as to minimise the error. Steps 5 and 6 above are referred to as the "**Backward Propagation Phase**".

7. Go back to step 1 until a satisfactory configuration is found.

3.5.1 Weight and Bias Update Expression for a Traditional Neural Network with a Variable Bias term

The network error E is defined as follows:

$$E = \frac{1}{2}\sum (t_k - o_k(o_j, w_{jk}))^2 \quad (3.11)$$

For output units $\frac{\partial E}{\partial w_{jk}}$ needs to be calculated where as for hidden units $\frac{\partial E}{\partial w_{ij}}$ is required. The respective weights would then be updated with the following equations.

$$\Delta w_{jk} = \eta \left(-\frac{\partial E}{\partial w_{jk}}\right) \tag{3.12}$$

$$\Delta w_{ij} = \eta \left(-\frac{\partial E}{\partial w_{ij}}\right) \tag{3.13}$$

The Calculation of $\frac{\partial E}{\partial w_{jk}}$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial w_{jk}} \tag{3.14}$$

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{3.15}$$

For the sigmoidal activation function,

$$o_k = \frac{1}{1+e^{-c_k \sum w_{jk} o_j}} \tag{3.16}$$

$$\frac{\partial o_k}{\partial w_{jk}} = -\left(\frac{1}{1+e^{-c_k \sum w_{jk} o_j}}\right)^2 (-c_k)\ (o_j)\ e^{-c_k \sum w_{jk} o_j}$$

$$= o_k{}^2 c_k o_j \frac{1-o_k}{o_k}$$

$$= o_k (1-o_k) c_k o_j \tag{3.17}$$

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) o_k (1-o_k) c_k o_j \tag{3.18}$$

Therefore, the weight update expression for the links connecting to output nodes with a bias is:

$$\Delta w_{jk} = \eta\, (t_k - o_k) o_k (1-o_k) c_k o_j \tag{3.19}$$

Similarly, the bias update expressions for the output nodes would be:

$$\Delta \theta_k = \eta\, (t_k - o_k) o_k (1-o_k) c_k \tag{3.20}$$

The Calculation of $\frac{\partial E}{\partial w_{ij}}$

The calculation of $\frac{\partial E}{\partial w_{ij}}$ is not very simple as the error depends on the output of hidden as well as output units. Therefore,

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial\left[\frac{1}{2}\sum(t_k - o_k)^2\right]}{\partial w_{ij}} \tag{3.21}$$

$$= \frac{\partial\left[\frac{1}{2}\sum(t_k - o_k)^2\right]}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} \tag{3.22}$$

The subscript j and k stand for hidden and output nodes respectively. To avoid the confusion while using other subscripts, superscripts h and o would be mentioned where necessary to denote for hidden and output nodes respectively.

Let's calculate $\frac{\partial\left[\frac{1}{2}\sum(t_k - o_k)^2\right]}{\partial o_j}$ term in Equation 3.22 is calculated as follows:

$$\frac{\partial\left[\frac{1}{2}\sum\left(t^o{}_k - o^o{}_k\right)^2\right]}{\partial o^h{}_j} = \frac{\partial\left[\frac{1}{2}\sum\left(t^o{}_1 - o^o{}_1\right)^2\right]}{\partial o^o{}_1}\frac{\partial o^o{}_1}{\partial o^h{}_j} + \dots$$

$$= \dots + \frac{\partial\left[\frac{1}{2}\sum\left(t^o{}_k - o^o{}_k\right)^2\right]}{\partial o^o{}_k}\frac{\partial o^o{}_k}{\partial o^h{}_j} + \dots$$

$$= \dots + \frac{\partial\left[\frac{1}{2}\left(t^0{}_n - o^0{}_n\right)^2\right]}{\partial o^0{}_n}\frac{\partial o^o{}_n}{\partial o^h{}_n}$$

(3.23)

From equation 3.16,

$$\frac{\partial o^o{}_k}{\partial o^h{}_j} = -\left(\frac{1}{1+e^{-c_k}\sum w_{jk}o_j}\right)^2\left(-c_k w_{jk}\right)e^{-c_k\sum w_{jk}o^h{}_j}$$

$$= o_k{}^2 c_k w_{jk}\frac{1-o_k}{o_k}$$

$$= o_k\left(1-o_k\right)c_k w_{jk} \tag{3.24}$$

Now,

$$\frac{\partial\left[\frac{1}{2}\sum(t_k - o_k)^2\right]}{\partial o^o{}_k} = -\left(t^o{}_k - o^o{}_k\right) \tag{3.25}$$

Therefore, Equation 3.23 becomes,

$$\frac{\partial\left[\frac{1}{2}\sum(t_k - o_k)^2\right]}{\partial o_j} = -\sum_k c_k w_{jk} o_k (1 - o_k)(t_k - o_k) \tag{3.26}$$

Now, let's calculate the term $\frac{\partial o_j}{\partial w_{ij}}$

$$\frac{\partial o_j}{\partial w_{ij}} = -\left(\frac{1}{1 + e^{-c_j \sum w_{ij} o_i}}\right)^2 \left(-c_j o_i\right) e^{-c_j \sum w_{ij} o_i}$$

$$= o_j{}^2 o_i \frac{1 - o_j}{o_j} c_j$$

$$= c_j o_j (1 - o_j) o_i \tag{3.27}$$

Therefore, for each hidden unit j, substituting equations 3.26, 3.27 in Equation 3.21 we get,

$$\frac{\partial E}{\partial w_{ij}} = -\left[\sum_k c_k w_{jk} o_k (1 - o_k)(t_k - o_k)\right] c_j o_j (1 - o_j) o_i \tag{3.28}$$

Therefore, the weight update expressions for the links connecting to hidden nodes is:

$$\Delta w_{ij} = \eta \left[\sum_k c_k w_{jk} o_k (1 - o_k)(t_k - o_k)\right] c_j o_j (1 - o_j) o_i \tag{3.29}$$

Similarly, the bias update expressions for the hidden nodes would be:

$$\Delta \theta_j = \eta \left[\sum_k c_k w_{jk} o_k (1 - o_k)(t_k - o_k)\right] c_j o_j (1 - o_j) \tag{3.30}$$

3.6 Factors Affecting the Network Performance

Codes for traditional single and multi layer neural networks are developed by the author, and are written using C++ programming language, for supervised training using the back-propagation algorithm described in Section 3.5. The network parameters such as weights and biases for hidden and output nodes are initialised using small random numbers and updated using the weight and bias update expressions derived in Section 3.5.1. The effect of different parameters on network training is studied.

1. **Mode of Operation:** One complete presentation of the entire training set during a learning process is called an epoch. The learning process is iterated on an epoch-by-epoch basis until the synaptic weights and bias levels of the network stabilize and the average squared error over the entire training set converges to an acceptable low value. For a given training set, back-propagation learning may proceed in one of two following ways:

    - **Sequential Mode:** The sequential mode of back-propagation learning is also referred to as online mode. In this mode of operation, weight updating is performed after presentation of each training example.

    - **Batch Mode:** In the batch mode of back-propagation learning, weight updating is performed after presenting the entire set of training examples that constitute an epoch.

    Batch mode training usually requires less number of epochs for training, and is also less likely to give oscillations in weights as training progresses, however, the sequential mode of back-propagation learning is popular (particularly for solving pattern classification problems) for the following reasons:

- The algorithm is simple to implement.
- Coupled with random pattern selection strategies, it allows a weight correction that is somewhat random in nature.

- From an "on-line" operational point of view, sequential mode is preferred over batch mode because it requires less local storage for each synaptic connection and may become computationally faster than the batch mode if there is a high degree of redundant information in the data set. As a simple example, suppose that we create a larger training set from the original one simply by replicating the original data set ten times. In this case, the contribution to error in both nodes will be different and batch mode algorithm will require longer time to find a solution. By contrast, the sequential algorithm updates the weights after each pattern presentation, and so will be unaffected by the replication of data.
  However, training with batch mode becomes convenient when using more efficient optimization methods as compared with gradient descent method. Now the effect of both these modes on training is studied.

Consider a network (Figure 3.6), with two input nodes and one output node. The network is trained in sequential as well as batch modes for the same randomly chosen initial weight and bias values:

w_1 = -0.171131, w_2 = -0.863562 and bias b_1= -0.510387 for the output node.

Note that batch mode trains the network in 16 epochs as compared to the sequential mode, which trains in 477 epochs as indicated in Figures 3.19(a) and 3.19(b) below.

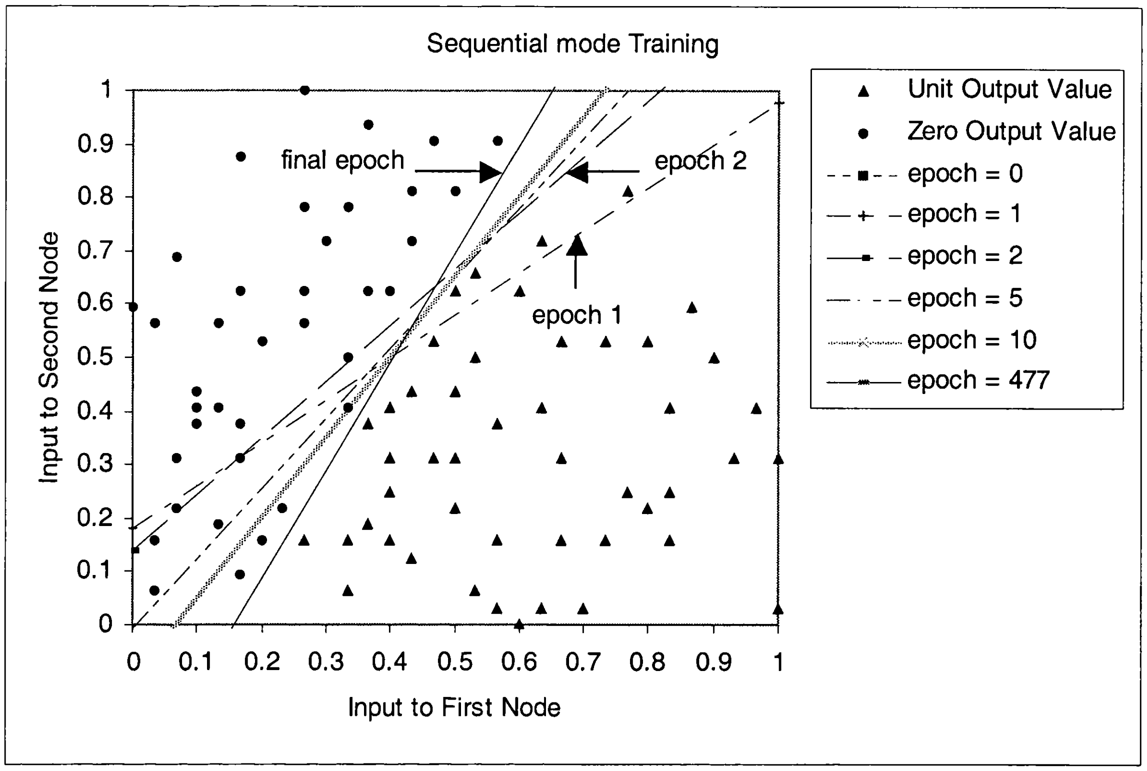


Figure 3.19(a) Sequential mode

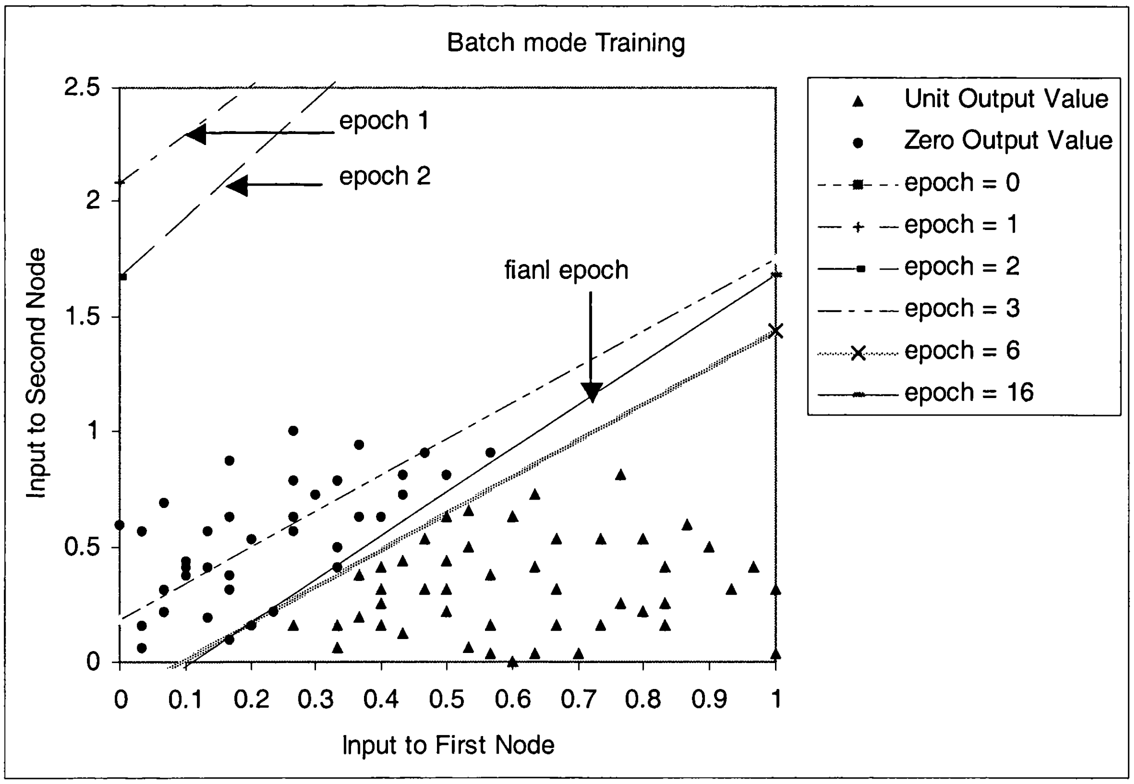


Figure 3.19(b) Batch mode

Figure 3.19: Training a network using mode (a) and mode (b) as above.

2. Learning Rate

The effect of varying learning rate has been shown for the same network discussed in the previous paragraph. The network is trained over the training example set, starting with the same set of initial weights and biases, for five different learning rates. The results of the experiment are compared directly. The learning curves so computed are plotted in Figure 3.20 below. It can be observed that increasing the learning rate results in faster training and decreasing the learning rate decreases the speed of training.

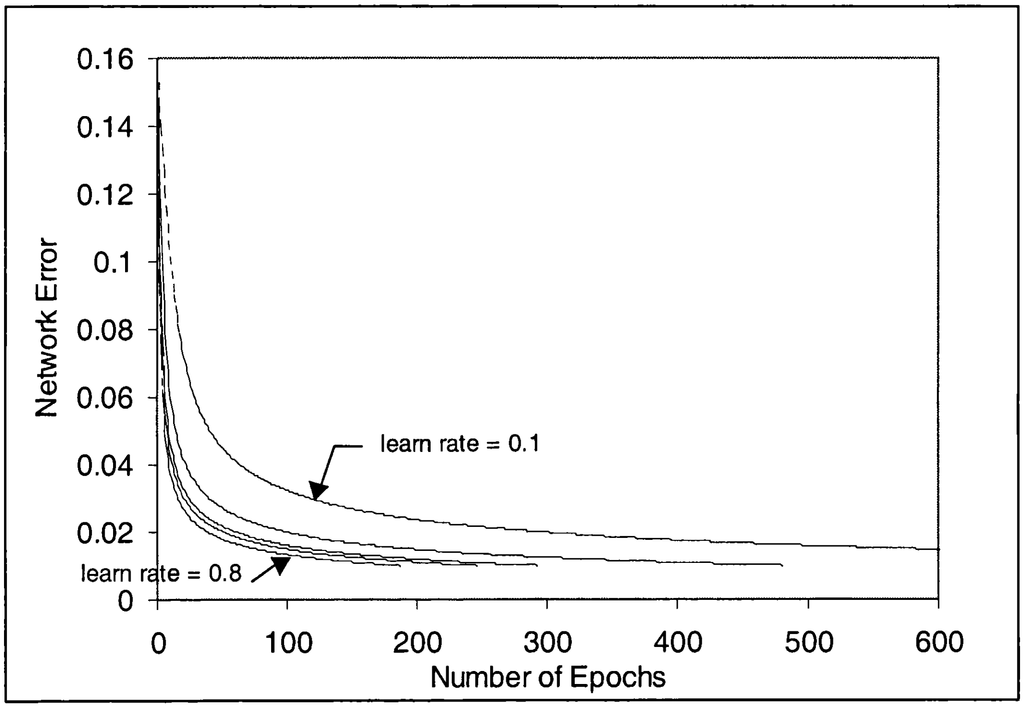


Figure 3.20 Effect of Different Learning Rates (i.e. 0.1, 0.3, 0.5, 0.6, 0.8) on Network Training

The learning rate determines the magnitude of weight change along the chosen direction in weight space. Most of the publications of neural network use a constant value between 0.3 and 0.6. If a larger value is chosen for the learning rate, it may speed up the learning process, however, may also result in large changes in the synaptic weights that make the network unstable. Smaller values of learning rates may avoid oscillatory changes in weights, however, at the cost of excessive computational time. A general rule is to use the largest learning rate that works and does not cause oscillation.

3. **Momentum Constant:** Another technique for improving the learning speed without introducing divergent oscillations in weight changes is to modify the weight update expression by including a momentum term [8]. A fraction of the weight change in the previous iteration is added to the weight change in the current iteration. This ensures a smooth change in weights by minimizing oscillations leading to a better convergence rate. The momentum constant, μ, is a fraction which determines the proportion of weight change in the previous iteration that is added in the current weight change. The incorporation of a momentum term in the back-propagation algorithm modifies the weight and bias update expressions for output as well as hidden nodes as follows:

The weight update expression for a link connecting j^{th} hidden node to k^{th} output node is:

$$\Delta w_{jk}(n+1) = \eta\,(t_k - o_k)\,o_k(1-o_k)\,c_k o_j + \mu\,\Delta w_{jk}(n) \tag{3.31}$$

Similarly, the bias update expression for k^{th} output node is:

$$\Delta \theta_k(n+1) = \eta\,(t_k - o_k)\,o_k(1-o_k)\,c_k + \mu\,\Delta \theta_k(n) \tag{3.32}$$

The weight update expression for a link connecting i^{th} input node to j^{th} hidden node is:

$$\Delta w_{ij}(n+1) = \eta\left[\sum_k c_k w_{jk} o_k(1-o_k)(t_k - o_k)\right] c_j o_j(1-o_j)\,o_i + \mu\,\Delta w_{ij}(n) \tag{3.33}$$

Similarly, the bias update expression for the hidden nodes would be:

$$\Delta \theta_j(n+1) = \eta \left[\sum_k c_k w_{jk} o_k (1-o_k)(t_k - o_k) \right] c_j o_j (1-o_j) + \mu \Delta \theta_j(n) \quad (3.34)$$

Although, the inclusion of momentum term leads to an improvement in the performance of the back-propagation algorithm, the algorithm still remains relatively inefficient as compared to more efficient optimisation methods. The inclusion of momentum term also introduces a second parameter 'μ', whose value needs to be chosen, in addition to that of the learning rate parameter η. Several attempts have been made to explicate the influence of momentum term upon the back propagation [3] to suggest proper choice of the term [4-5]. In addition to the learning rate parameter, momentum term also greatly influences the convergence speed of the training procedure. As discussed before, the larger the learning rate η, the larger the change in the weights. However, by using a moderate value of the learning rate and adding a momentum term filters out the oscillations [6, 9]. A relationship between the momentum constant μ, learning rate η and learning speed in epochs has been reported in literature [3,4,7] that the learning speed is proportional to $\left(\frac{1-\mu}{\eta} \right)$. Attoh-Okine [10] showed that learning rate of around 0.2 to 0.5 and momentum term of around 0.4-0.5 provides the appropriate combination rather than very small learning rate, roughly 0.001 and a relatively high momentum term between 0.5-0.9. It has also been observed that learning rate and momentum constant have a significant impact on the training speed, but not on the generalization ability [5].

3.7 Conclusion

Basic principles of neural networks have been introduced in this chapter. The network architecture comprises of input, hidden and output nodes arranged in appropriate layers. The processing at each neuron is almost the same and the final network architecture is generally determined by a trial and error method.

The number of hidden layers and also hidden nodes in each layer determine the non-linearity of a mapping function in input and output space. The generalisation ability is directly determined by the number of hidden nodes in the network.

This chapter forms a foundation for some of the advanced concepts introduced in later chapters. Codes for single and multi layer feed forward neural networks have been developed by the author using C++ programming language.

REFERENCES

[1] Philip D. Wasserman, Van Nostrand Reinhold, Neural Computing Theory and Practice, New York, 1989, ISBN-0-442-20743-3.

[2] Simaon Haykin, Neural Networks A Comprehensive Foundation, Second Edition, Prentice Hall, New Jersey, 1999, ISBN-0-13-273350-1.

[3] Perantonis S.J., Karras D.A., "An Efficient Constrained Learning Algorithm With Momentum Acceleration", *Neural Networks*, **vol** 8, **no** 2, 1995, **pp** 237-249.

[4] Sato A., "An Analytical Study of the Momentum Term in a Back Propagation Algorithm", *Proceeding of ICANN91*, 1991, **pp** 617-622.

[5] Holger R.M., Graeme C.D., "The Effect of Internal Parameters and Geometry on the Performance of Back-Propagation Neural Networks", *Environmental Modelling and Software,* vol **13**, no **1**, 1998, **pp** 193-209.

[6] Dai H., MacBeth C., "Effects of Learning Parameters on Learning Procedure and Performance of a BPNN", *Neural Networks,* **vol 10**, **no** 8, 1997, **pp** 1505-1521.

[7] Hagiwara M., Sato M., Sato A., "Analysis of Momentum Term in Back-Propagation", *IEICE Transactions on Information and Systems*, **vol** E78-D, **no** 8, August 1995, **pp** 1080-1086.

[8] Looney C.G., "Stabilization and Speedup of Convergence in Training Feed Forward Neural Networks", *Neurocomputing*, **vol** 10, **no** 1, 1996, **pp** 7-31.

[9] Qian N., "On the Momentum Term in Gradient Descent Learning Algorithms", *Neural Networks*, vol **12**, 1999, pp **145-151**.

[10] Attoh-Okine N.O., "Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance", *Advances in Engineering Software*, vol **30**, 1999, pp **291-302**.

CHAPTER 4

Effect of Gain on the Learning Abilities of Neural Networks

The slope of the activation function determines the range over which output values are changed from zero to one for a given set of weighted inputs. In a 'feed forward' algorithm, this slope is directly influenced by a parameter referred to as 'gain'. A unit value for gain has generally been used for most of the research reported in the literature. In this chapter, the influence of the variation of 'gain' on the learning ability of a neural network is analysed. The chapter begins with an introduction to gain and its geometric interpretation. Applications to both single layer and multi layer neural networks have been assessed.

During the work, it was discovered that 'gain' is not an independent parameter as perceived before undertaking this study. The relationship between network weights, learning rate and gain has also been highlighted. Although, it was not possible to extend the geometric interpretation of a 'gain' value to advance the learning abilities of the network to associate causes with observed effects, it is shown that the training time can be significantly reduced by changing the 'gain' value adaptively for each node.

4.1 Introduction

The calculations performed during the forward phase of a feed forward neural network algorithm are described in Chapter 3. As shown in Equation 4.1, the weighted sum of the inputs is passed through a sigmoid activation function to generate the nodal output (Refer to Equation 3.1 for details).

$$\text{Output of the } j^{th} \text{ node } o_j = \frac{1}{1+e^{-c_j a_{net,j}}} \tag{4.1}$$

$$\text{where, the net input } a_{net,j} = \sum_i w_i o_i + \theta_j$$

The activation function is graphically shown in Figure 4.1. The value of the gain parameter directly influences the slope of the activation function. For large gain values (c >> 1), the activation function approaches a 'step function' (Figure 3.12, Chapter 3) whereas for 0 < c << 1, the output values change from zero to unity over a large range of the weighted sum of the input values. Before undertaking this study [1], it was felt that this particular feature of gain variation might influence the way input values are mapped to the output values. This was considered important because output values of diagnostic neural networks – which are the focus of this study – range from zero to unity. The following two diagnostic cases further support this argument.

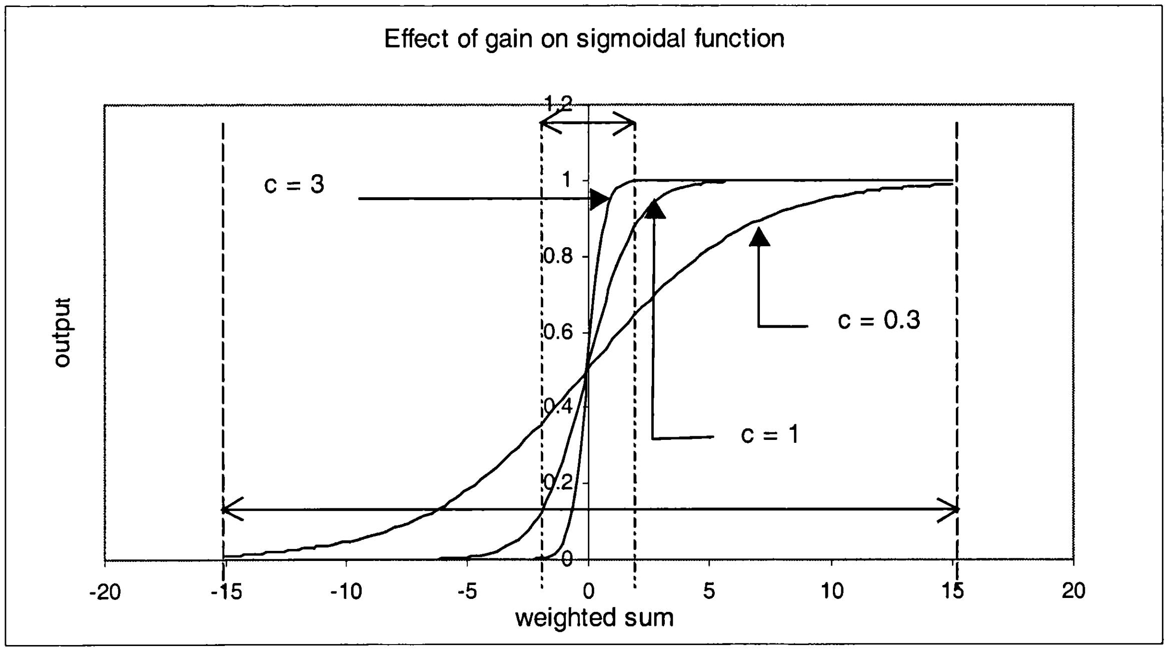


Figure 4.1 Sigmoid activation function with different slopes

Case 1: A cause is associated with two defect nodes in a single layer network as shown in Figure 4.2.

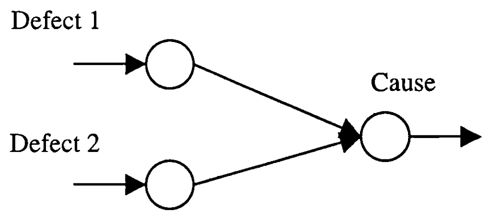


Figure 4.2 Simple network connecting two 'defect' nodes to a 'cause' node.

Case 2: A cause is associated with two defect nodes in a multi layer network (one hidden layer and two hidden nodes) as shown in Figure 4.3.

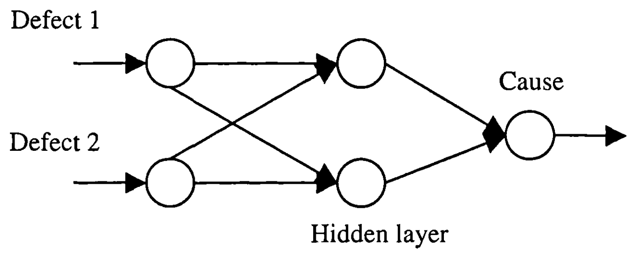


Figure 4.3 Multi layer network connecting two 'defect' nodes to a 'cause' node.

As described in Chapter 2, for diagnostic networks, the input value represents the relative strength of occurrence of a defect in the rejection data and the output value represents the belief in the occurrence of a cause for a given relative strength of occurrences of both defects.

The interpretation of the output value is rather different from many of the traditional neural network applications, where output nodes generally have binary output values. In other words, the output of the diagnostic network is a vector of outputs with values ranging from 0 to 1. These output values are classified into three regions, referred to as A, B and C (Figures 4.4 and 4.6).

- Region A is the region of zero output values.
- Region B shows output values ranging from 0 to 1.

- Region C has unit output values.

Figure 4.4 shows the classification of input patterns undertaken by the network shown in Figure 4.2. Similarly, Figure 4.6 shows the classification of input patterns undertaken by the multi layer network shown in Figure 4.3. In this case, the decision boundaries which separate the three regions are curves (Figure 4.6). The methodology used to plot these lines and curves is derived in Appendix 4A and 4B at the end of this chapter.

Figure 4.5 presents a detailed classification of output values shown in Figure 4.4. Region B, as shown in Figure 4.4, is termed as linearly separable because the boundaries of Region B are straight lines. Similarly, the classification of output values shown in Figure 4.6 is detailed in Figure 4.7. Hidden nodes, and the hidden layer, have introduced the non-linearity (curvature) in the decision boundary.

It can be observed in Figures 4.5 and 4.7 that the output values in the Region B vary exponentially from the central output value equal to 0.5. For the diagnostic problem under study, it was felt that the region of interest, where most of the training data values lie, would be Region B, which is a continuous space of output values between 0 and 1. By increasing or decreasing the width of region B, a greater number of network output values could be matched with the given output values. The alteration of this region is achieved by changing the gain value.

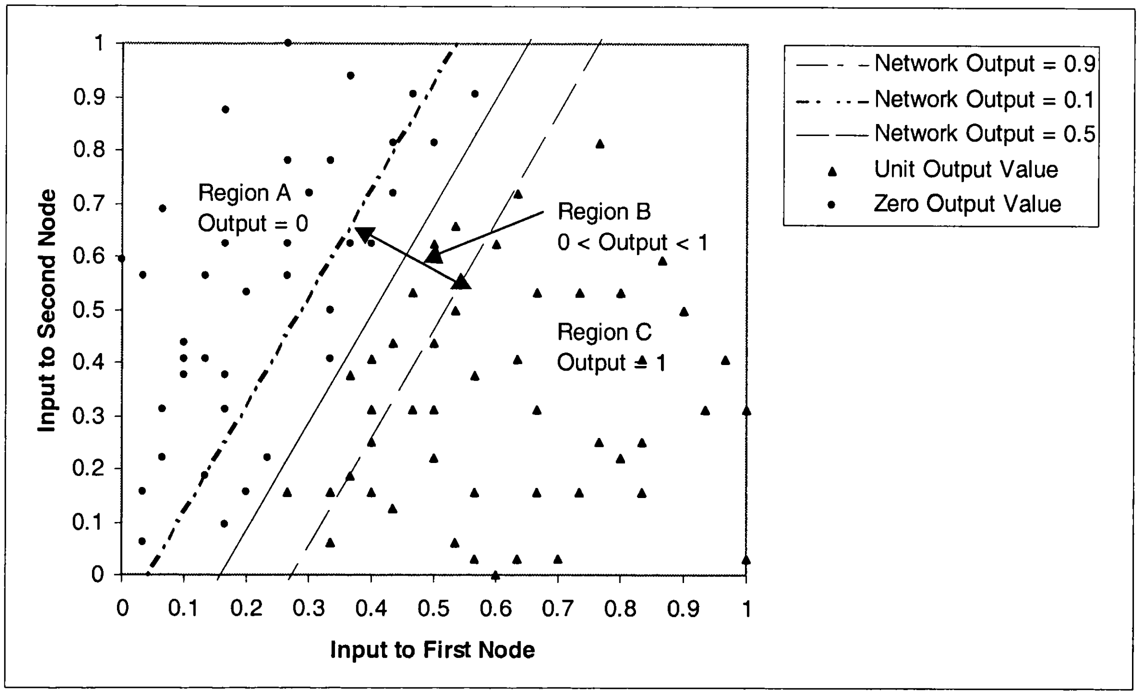


Figure 4.4 Classification of Input Pattern for Single layer networks

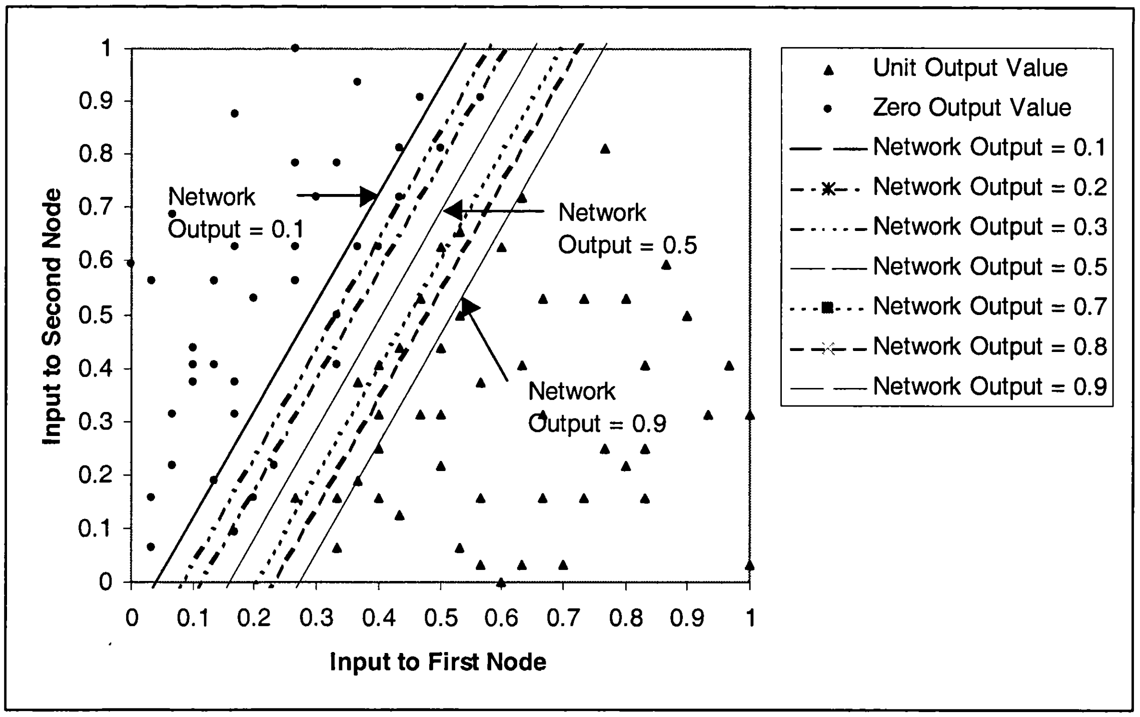


Figure 4.5 Detailed Classification of Input Pattern for Single layer networks

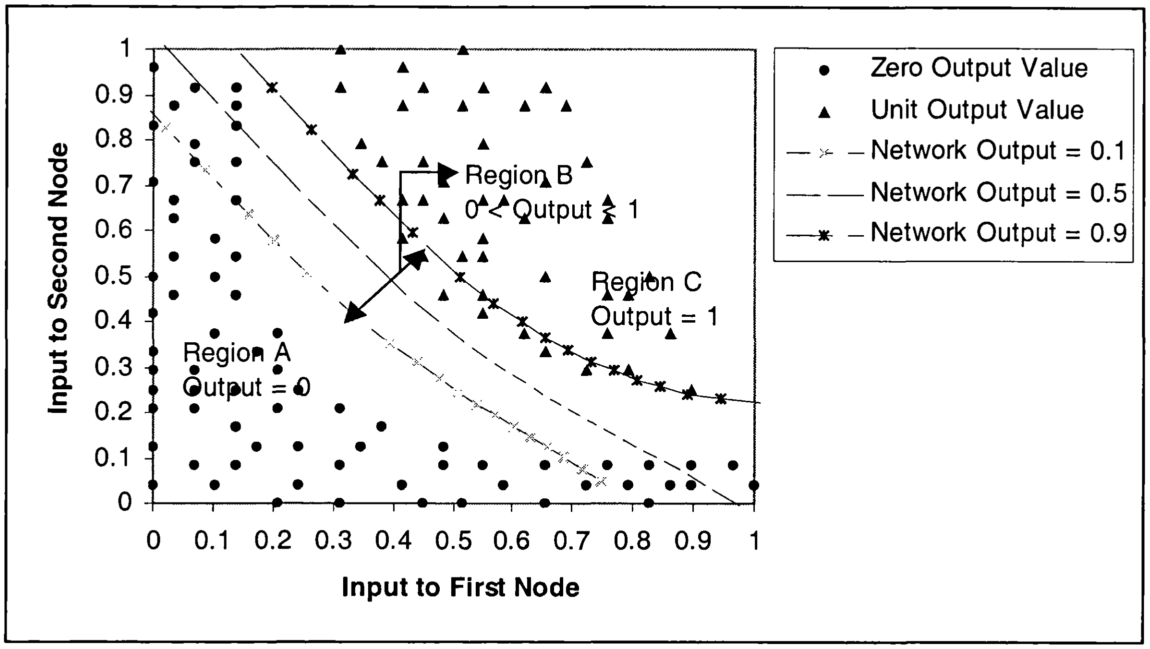


Figure 4.6 Classification of Input Pattern for Multi-layer networks

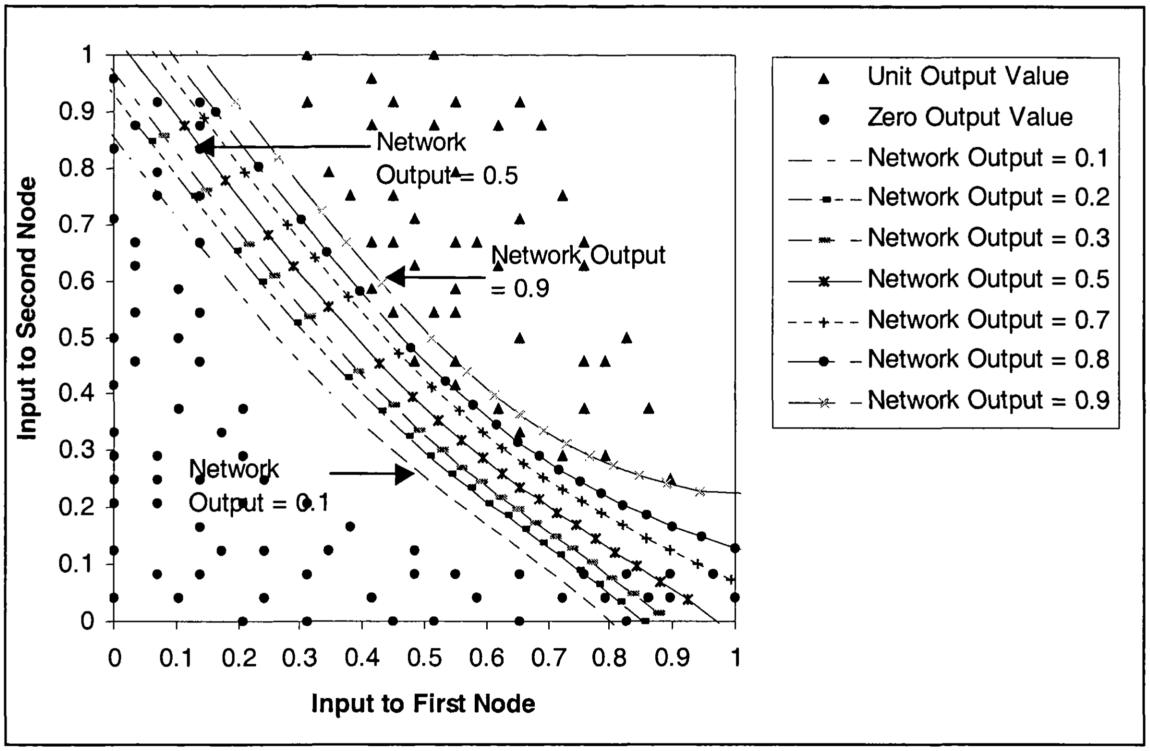


Figure 4.7 Detailed Classification of Input Pattern for Multi-layer networks

4.2 Influence of the Gain Value on the Classification Abilities of the Neural Network

The same case studies, (Figures 4.2 and 4.3), are studied further to analyse the influence of different values of gain during the training as well as the testing phase.

4.2.1 Influence of the Gain Value on Network Testing

Example 1: Consider the single layer network as shown in Figure 4.2. An input pattern is presented to this network and the network is trained with a unit gain value. Then it is tested for different gain values such as c = 3 and c = 1/3 = 0.33. The decision boundaries are plotted for an output value equal to 0.1, 0.5 and 0.9. The region B, which is a continuous space of output values between zero and one is shown between the decision lines for an output value equal to 0.1 and an output value equal to 0.9. The effect of different gain values on this region is observed and plotted in Figure 4.8. The following observations are made.

- Increasing the gain value decreases the width of the region of values between 0 and 1.
- Decreasing the gain value increases the width of the region of values between 0 and 1.
- For all values of gain the straight line for output equal to 0.5 is the same.
- The equation of the line for an output equal to 0.5 is derived in Appendix 4A at the end of this chapter.
- The slope of the lines representing other output values such as 0.1, 0.5 and 0.9, remain the same but their intercepts are different (Figure 4.8). (Refer to Appendix 4A at the end of this chapter.)

This also holds for an n-dimensional network i.e. a network with n input nodes.

Example 2: For the multi-layer network, as shown in Figure 4.3, we have the same set of observations (Figure 4.9). The only difference is that the decision boundary is non-linear.

Both single and multi layer networks are trained with a unit gain value. These networks predict fractional output values for only a small part of the training data due to the narrow region B. The performance of the network on different values of gain during a testing phase is analysed (Figures 4.8 and 4.9), and the following observations are made:

- With a sufficiently large region B, which can be achieved by decreasing the value of gain, the simple network can also predict fractional output values for most of the training data.

- However, the few data points that are still left outside this region have only binary output values.

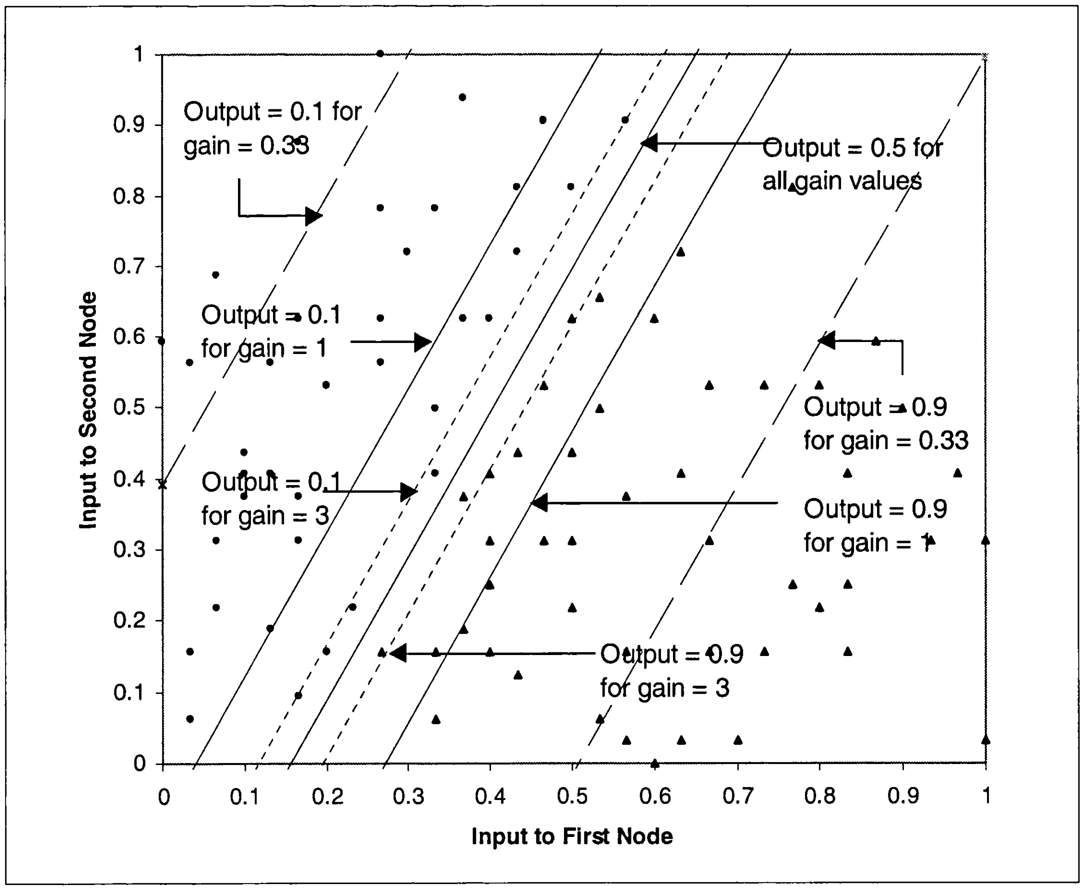


Figure 4.8 Testing the single layer network for different values of gain

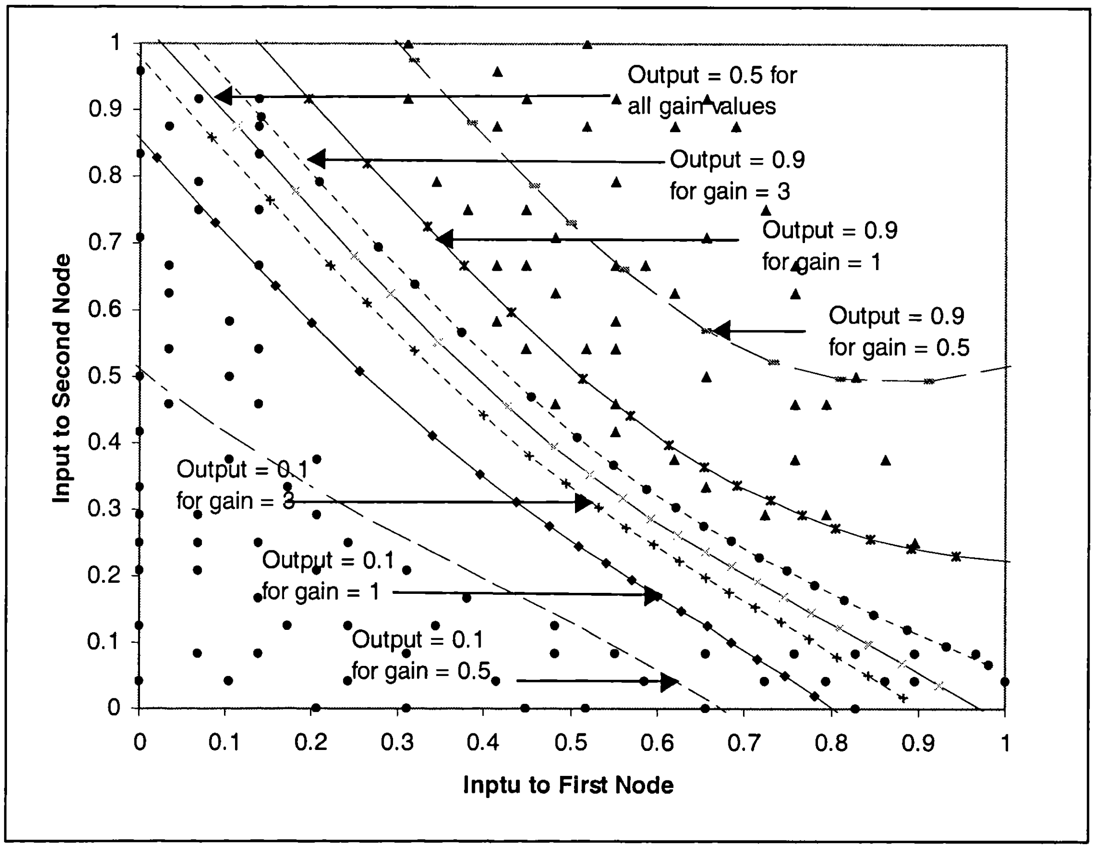


Figure 4.9 Testing the multi-layer network for different values of gain

4.2.2 Influence of the Gain Value on Network Training

Example 1: The single layer network, as shown in Figure 4.2, is now trained for different values of gain such as c = 3, and c = 0.3. The decision boundaries are again plotted for output values equal to 0.1, 0.5 and 0.9 as shown in Figures 4.10 and 4.11. The following observations are made.

- Increasing the gain value decreases the width of the region of values between 0 and 1.
- Decreasing the gain value increases the width of the region of values between 0 and 1.
- The widths of regions of fractional output values for different training sets are different.

- The slopes of decision boundaries for different training sets are also different.
- Within a training set, the decision boundary for an output equal to 0.5 is the same for all values of gain.
- Within a training set, the slopes of the boundaries for all output values are also the same, but their intercepts are different.

After the network was trained for different gain values, the results file was studied and an important observation is made as follows:

As the gain value was increased, the error was reduced in a fewer number of epochs, and, the training process required a greater number of epochs as we decreased the value of gain. The training process required 4209 epochs to achieve the target error of 0.01 with a gain value of 0.3 whereas the same network was trained in 64 epochs with a gain value of 3. This is illustrated in Figure 4.12. In other words, it was observed that the gain and learning rate influence the learning performance of the network in a similar way.

For the purpose of simplicity, the single layer network as shown in Figure 4.2 has been chosen and the effect of different values of gain on the training process has been analysed. From Figures 4.10 and 4.11, it can be observed that different values of gain used during the training process do not really achieve a desired positive control on all the output values. The width of the region B is decided by the final weights. Also, we still need to test the network by modifying the region B. Nevertheless, the value of gain used during training influences the rate of learning. Therefore, it was then decided to investigate whether the change in a gain value has the same effect as changing the learning rate.

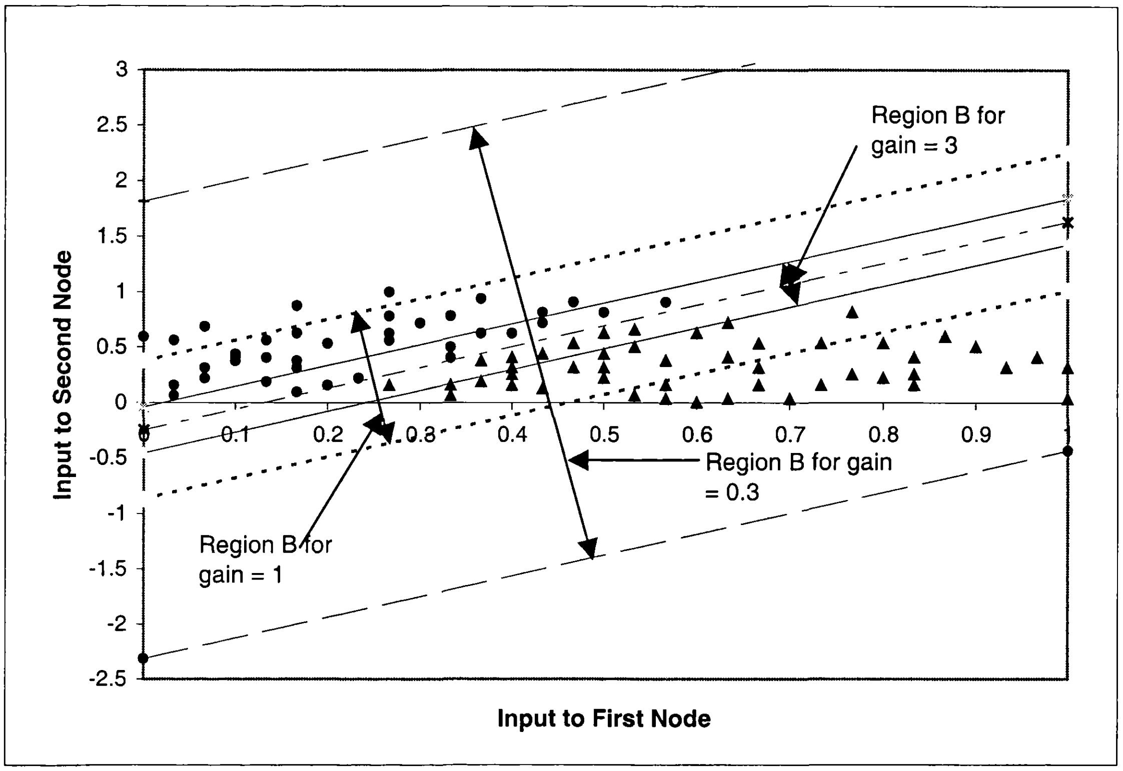


Figure 4.10 Training the network with gain = 3

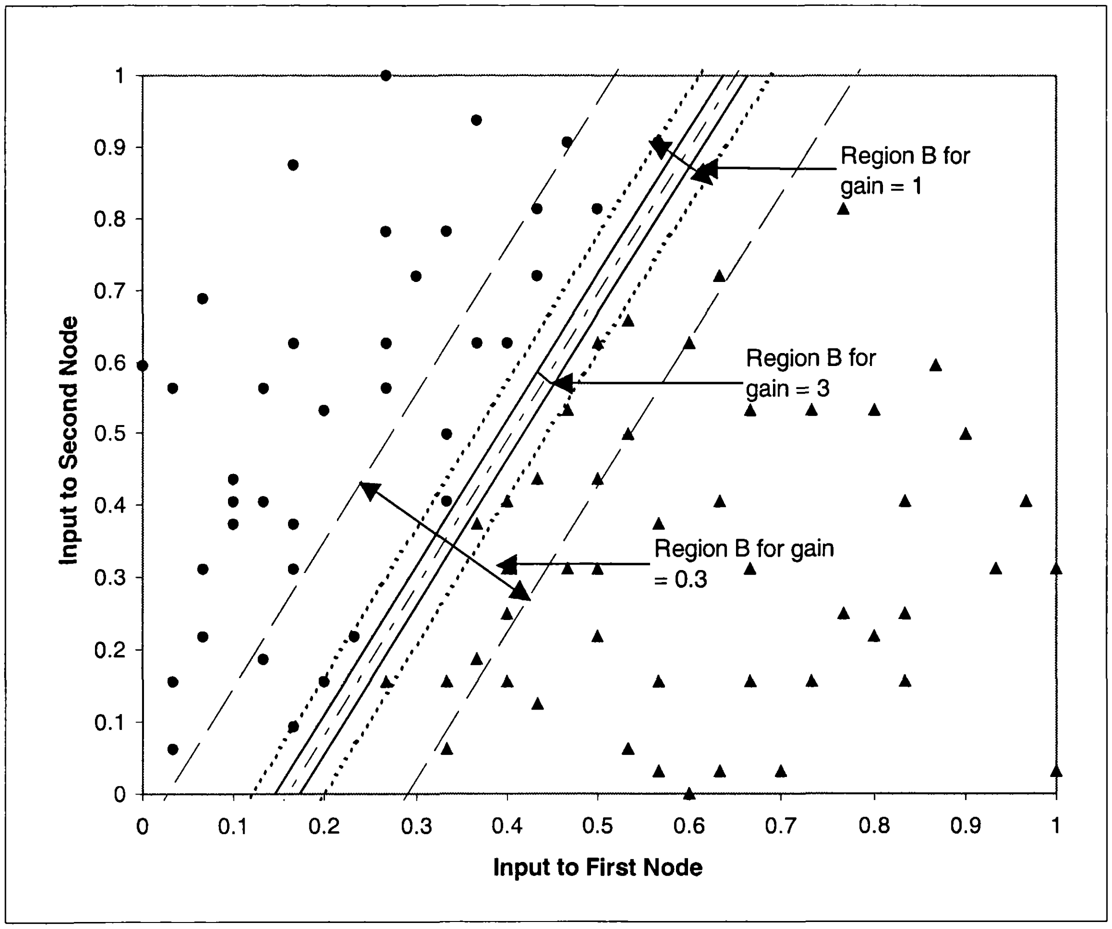


Figure 4.11 Training the network with gain = 0.3

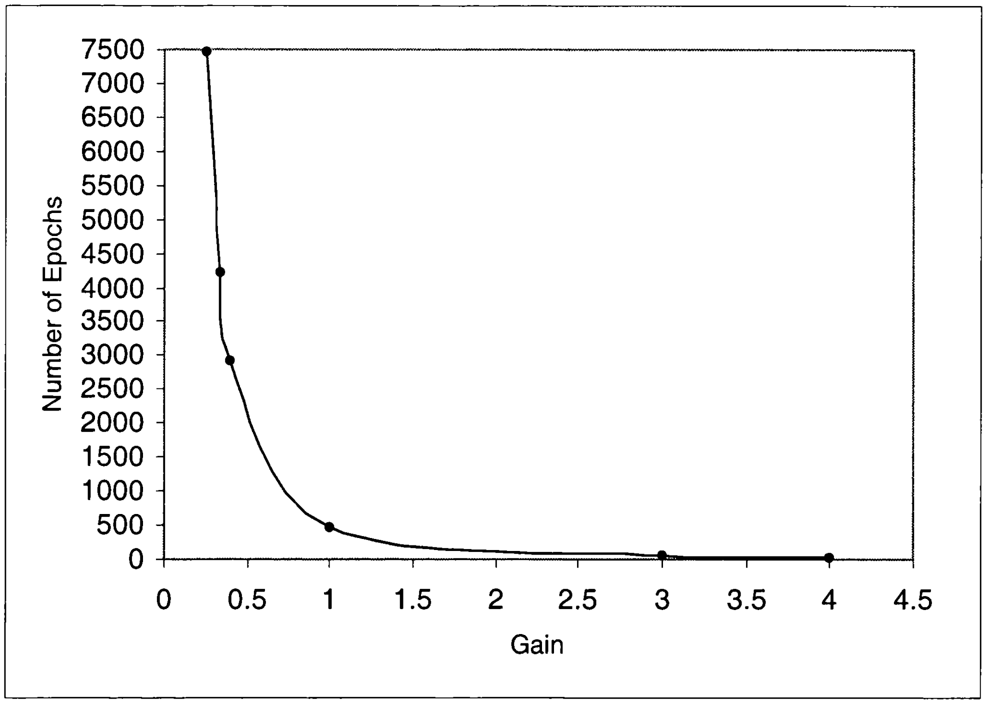


Figure 4.12 Effect of gain on Network Training

4.3 Relationship between the Gain Value, the Learning Rate, and the Initial Weight Values

A relationship between the gain value, a set of initial weight values, and a learning rate value has been described in this section and shown for two back-propagation neural networks X and Y with identical topology as shown in Figures 4.13 and 4.14. The neurons in the networks have activation functions a_j and $\overline{a}_j$ respectively and Network X has gain $c \neq 1$ and Network Y has $\overline{c} = 1$.

4.3.1 Calculation Procedure

1. **Initialisation:** Weights and biases are initialised randomly but the same set of initial weights and biases is used for both networks. However, for the second network (Y), the initial weights as well as the biases are multiplied by the gain value 'c' of the first network (X).

2. **Pattern presentation:** Training data containing an input pattern with its corresponding target pattern is presented in the same order for training both networks.

3. The learning rate $\overline{\eta}$ of the second network is the learning rate η of the first network multiplied by a factor c^2.

The two networks X and Y are as follows. We will consider a jth node of both networks to prove the relationship.

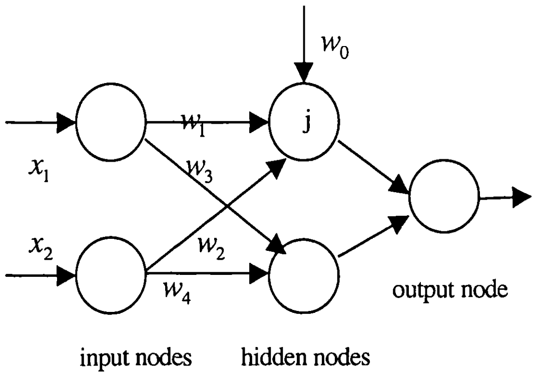


Figure 4.13: Network X

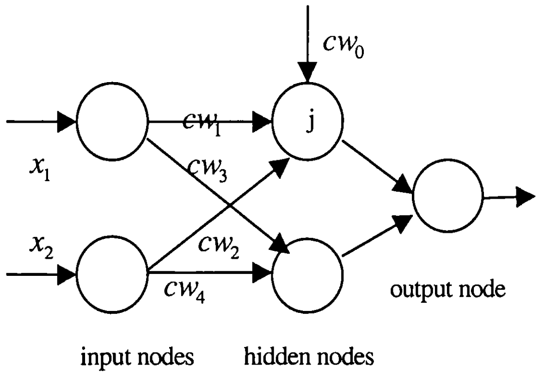


Figure 4.14: Network Y

Network X

weights	$w = [w_1, w_2, w_3, w_4]$
gain	$c \neq 1$
learning rate	η
bias	w_0

Output :

$$a_j = \frac{1}{1 + e^{-ca_{net,j}}}$$

where,

weighted sum

$$a_{net,j} = x_1 w_1 + x_2 w_2 + w_0 \qquad (4.2)$$

Network Y

weights	$\overline{w} = cw$
gain	$\overline{c} = 1$
learning rate	$\overline{\eta} = c^2 \eta$
bias	cw_0

Output :

$$\overline{a}_j = \frac{1}{1 + e^{-\overline{c}\,\overline{a}_{net,j}}}$$

where,

weighted sum

$$\overline{a}_{net,j} = x_1 cw_1 + x_2 cw_2 + cw_0$$

$= c(x_1 w_1 + x_2 w_2 + w_0)$

Substituting from Equation 4.2 we have,

$\overline{a}_{net,j} = c a_{net,j}$

$\therefore \; \overline{a}_j = \dfrac{1}{1 + e^{-c a_{net,j}}}$

$$\underline{\therefore a_j = \overline{a}_j}$$

This shows that for both networks, the activation value in any hidden node is exactly the same. Since the activation value is considered to be equal to the output value, for the hidden layer we have $a_j = o_j$ and similarly, for the output layer it can be shown that $a_k = o_k$.

$$\underline{\therefore o_j = \overline{o}_j} \text{ and } \underline{o_k = \overline{o}_k} \qquad (4.3)$$

As per the assumption, the following equation holds for the initial weights of both networks:

Weights for the first iteration: w_1^{1} | $\overline{w}_1^{1} = c w_1^{1}$

If we assume that this relationship holds for the nth iteration of the training cycle, then to show that both networks are equivalent, we need to prove that the relationship also holds for the (n+1)th training iteration.

Weight Correction term: Δw_1 | $\Delta \overline{w}_1$

Weights for the (n+1)th iteration:

$w_1^{n+1} = w_1^{n} + \Delta w_1$ | $\overline{w}_1^{n+1} = \overline{w}_1^{n} + \Delta \overline{w}_1$

To prove that: if the weight correction term $\Delta \overline{w}_1 = c \Delta w_1$, then $\overline{w}_1^{n+1} = c w_1^{n+1}$ holds, consider the following:

$$\Delta w_1 = \eta * \frac{\partial E}{\partial w_1}$$

$$\Delta \overline{w}_1 = \overline{\eta} * \frac{\partial \overline{E}}{\partial \overline{w}_1}$$

For the output layer :

$$\Delta\, w_{jk} = \eta\, (t_k - o_k) o_k (1 - o_k) c o_j$$

$$\Delta w_{jk} = \eta \partial'_k\, o_j$$

$$\partial'_k = (t_k - o_k) o_k (1 - o_k) c$$

$$\Delta \overline{w}_{jk} = \overline{\eta}\, \overline{\partial}'_k\, \overline{o}_j$$

$$\overline{\partial}'_k = (t_k - \overline{o}_k) \overline{o}_k (1 - \overline{o}_k) \overline{c}$$

$\because o_j = \overline{o}_j$, and $\overline{c} = 1$, we have

$$\overline{\partial}'_k = (t_k - o_k)\ \ o_k (1 - o_k)$$

$$\therefore \partial'_k = \ \ c \overline{\partial}'_k \qquad (4.4)$$

For the hidden layer :

$$\Delta\, w_{ij} = \eta \left[\sum_k c w_{jk} o_k (1 - o_k)(t_k - o_k) \right] c o_j (1 - o_j) o_i$$

$$\Delta w_{ij} = \eta \partial'_j\, o_i$$

$$\partial'_j = \sum_k \partial'_k w_{jk}\, o_j (1 - o_j) c$$

$$\Delta \overline{w}_{ij} = \overline{\eta}\, \overline{\partial}'_j\, \overline{o}_i$$

$$\overline{\partial}'_j = \sum_k \overline{\partial}'_k \overline{w}_{jk}\, \overline{o}_j (1 - \overline{o}_j) \overline{c}$$

Substituting for $\overline{\partial}'_k, \overline{w}_{jk}$ and $\overline{c} = 1$,

$$\overline{\partial}'_j = \sum_k \frac{\partial'_k}{c}\ \ c w_{jk} o_j (1 - o_j)$$

$$\overline{\partial}'_j = \sum_k \partial'_k\ \ w_{jk} o_j (1 - o_j)$$

$$\therefore \partial'_j = c\ \ \overline{\partial}'_j \qquad (4.5)$$

For the output layer:

The weight correction term: $\Delta \overline{w}_{jk} = \overline{\eta} \overline{\partial}'_k \overline{o}_j$

Substituting from Equations (4.3) and (4.4) we have $\Delta\overline{w}_{jk} = c^2\eta \frac{\partial'_k}{c} o_j$

$$\Delta\overline{w}_{jk} = c\eta\, \partial'_k o_j$$

$$\therefore \Delta\overline{w}_{jk} = c\Delta w_{jk} \tag{4.6}$$

For the hidden layer:

Weight correction term: $\Delta\overline{w}_{ij} = \overline{\eta}\overline{\partial}'_j\overline{o}_i$

Substituting from Equations (4.3) and (4.5) we have $\Delta\overline{w}_{ij} = c^2\eta \frac{\partial'_j}{c} o_i$

$$\Delta\overline{w}_{ij} = c\eta\, \partial'_j o_i$$

$$\therefore \Delta\overline{w}_{ij} = c\Delta w_{ij} \tag{4.7}$$

From both Equation (4.6) and (4.7) we can prove that,

$$\Delta\overline{w} = c\Delta w \tag{4.8}$$

This relationship also holds when both networks have the same values of momentum constants. Some sample numerical calculations to demonstrate the relationship have been shown in Appendix 4D. The calculations demonstrate the following observations, which also confirm the relationship between initial weight values, the learning rate and the gain value.

1. For a gain value 'c' equal to 4, the final weight and bias values of Network 2 are four times the final weights and biases of network 1.
2. The derivative terms for both the hidden and output nodes of Network 1 are four times the derivative terms for both the hidden and output nodes of Network 2.
3. The output of both the networks is identical.
4. The decision boundary is also identical for both cases (Figure 4.15).

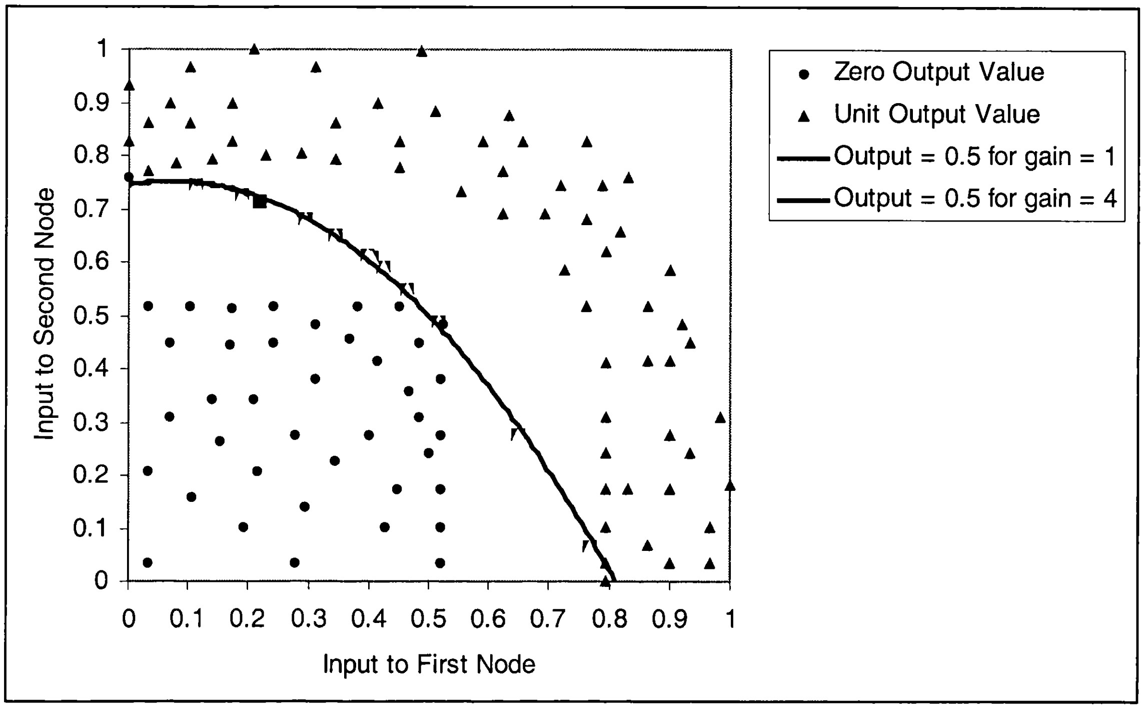


Figure 4.15 Non-linear Classification using Network 1 and Network 2 of Example 2, trained with different values of gain

4.4 Physical Interpretation of the Relationship between the Gain Value and Learning Rate

The effect of different values of gain on the slope of the sigmoid activation function is also analysed (Figure 4.16). The derivative of the output value with respect to the weighted sum $a_{net,j} = \left(\sum_{i=1}^{l} w_{ij} o_i \right)$ is calculated and plotted against the weighted sum. The expression for the derivative is,

$$\frac{\partial o_j}{\partial \sum_j w_{ij} o_i} = \frac{c e^{-c\left(\sum_j w_{ij} o_i + \theta_j\right)}}{\left(1 + e^{-c\left(\sum_j w_{ij} o_i + \theta_j\right)}\right)^2} \tag{4.9}$$

The weights are updated with the following equation.

$$\Delta w_{ij} = \eta \left(-\frac{\partial E}{\partial w_{ij}}\right) \tag{4.10}$$

The slope of the activation function controls the learning speed. For higher values of gain, the slope of the activation function increases thereby increasing the value of Δw_{ij} [6]. This results in faster training. Therefore, the type of activation function used has an influence on the learning speed, as the Gain value is different for various activation functions [3].

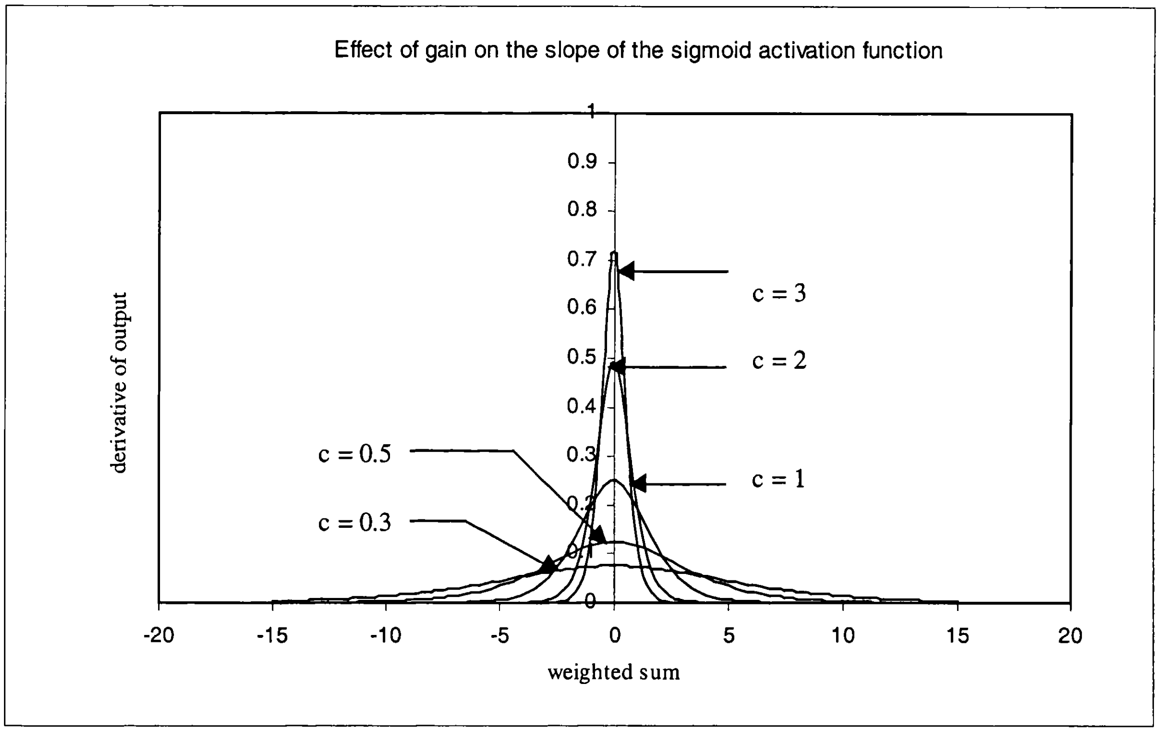


Figure 4.16 An analysis of the effect of gain on the slope of the sigmoid function

4.5 Physical Interpretation of Gain and Weight Values

Consider the sigmoid activation function as shown in Equation 4.1. The Gain parameter 'c_j' of the j^{th} node is a multiplying factor of the net input, $a_{net,j} = \sum_i w_i o_i + \theta_j$, of that node. This means that the results obtained by using a different value of gain are equivalent to those obtained by multiplying the net input by a factor equal to the gain value. The following observations are made.

1. The hyperbolic tangent activation function has a higher learning speed as compared to the logistic sigmoid activation function. This is because the gain value of the hyperbolic tangent activation function (Equation 4.11) is greater than that used in the logistic sigmoid activation function,

$$a_j = \tanh\left(a_{net,j}\right) = \frac{e^{ca_{net,j}} - e^{-ca_{net,j}}}{e^{ca_{net,j}} + e^{-ca_{net,j}}} \tag{4.11}$$

   In other words, a neural network whose hidden units use the activation function in Equation (4.11) is equivalent to one with hidden units using Equation (4.1), but having different values for the weights and biases.

2. A sigmoidal hidden unit approximates a linear hidden unit by using very small values of weights and biases. Similarly, a sigmoidal hidden unit can approximate a step function by setting the weights and bias feeding into that unit to very large values.

The above observations, coupled with the observations made in Section 4.3, show that, the same set of results can be achieved by either modifying the gain value or modifying the weights of the network.

4.6 Influence of Adaptive Gain on Network Training

Although the effect of different values of gain during training has been analysed in this chapter, these values are kept constant throughout the training process. In this section the advantages of using an adaptive gain value have been explored. Gain update expressions for output as well as the hidden nodes is derived in Section 4.6.1 below. For the sake of continuity, the weight, bias and gain update expressions for output as well as the hidden nodes are shown below.

The weight update expression for the links connecting to output nodes with a bias is:

$$\Delta w_{jk} = \eta\,(t_k - o_k)o_k(1 - o_k)c_k o_j \tag{4.12}$$

Similarly, the bias update expressions for the output nodes is:

$$\Delta \theta_k = \eta \left(t_k - o_k\right) o_k \left(1 - o_k\right) c_k \tag{4.13}$$

The gain update expression for links connecting output nodes is:

$$\Delta\ c_k = \eta\ \left(t_k - o_k\right)\ o_k \left(1 - o_k\right) \left(\sum_j w_{jk} o_j + \theta_k \right) \tag{4.14}$$

The weight update expressions for the links connecting to hidden nodes is:

$$\Delta\ w_{ij} = \eta \left[\sum_k c_k w_{jk} o_k \left(1 - o_k\right)\left(t_k - o_k\right) \right] c_j o_j \left(1 - o_j\right) o_i \tag{4.15}$$

Similarly, the bias update expressions for the hidden nodes is:

$$\Delta \theta_j = \eta \left[\sum_k c_k w_{jk} o_k \left(1 - o_k\right)\left(t_k - o_k\right) \right] c_j o_j \left(1 - o_j\right) \tag{4.16}$$

The gain update expression for the links connecting hidden nodes is:

$$\Delta\ c_j = \eta \left[\sum_k c_k w_{jk} o_k \left(1 - o_k\right)\ \left(t_k - o_k\right) \right]\ o_j \left(1 - o_j\right) \left(\sum_j w_{ij} o_i + \theta_j \right) \tag{4.17}$$

It can be observed that (Equations 4.15, 4.16 and 4.17) the weight, bias and gain update expressions are coupled. Earlier research [2-5] on using adaptive gain values has not used this coupling, and to the authors' knowledge such a coupled algorithm has been proposed for the first time. The algorithm has been proposed for sequential as well as batch training.

4.6.1 Mathematical Derivation to find an Expression for the Gain Change similar to Weight Changes during the Learning Stage

Here we calculate error with respect to the gain. The network error E is defined as follows:

$$E = \frac{1}{2}\sum (t_k - o_k(o_j, c_k))^2 \tag{4.18}$$

For output units, $\frac{\partial E}{\partial c_k}$ needs to be calculated whereas for hidden units. $\frac{\partial E}{\partial c_j}$ is also required. The respective gain values would then be updated with the following equations.

$$\Delta c_k = \eta\,(-\frac{\partial E}{\partial c_k}) \tag{4.19}$$

$$\Delta c_j = \eta\,(-\frac{\partial E}{\partial c_j}) \tag{4.20}$$

The calculation of $\frac{\partial E}{\partial c_k}$:

$$\frac{\partial E}{\partial c_k} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial c_k} \tag{4.21}$$

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{4.22}$$

For the sigmoidal activation function,

$$o_k = \frac{1}{1 + e^{-c_k \sum w_{jk} o_j + \theta_k}} \tag{4.23}$$

$$\frac{\partial o_k}{\partial c_k} = -\left(\frac{1}{1 + e^{-c_k \sum w_{jk} o_j}}\right)^2 e^{-c_k \sum w_{jk} o_j + \theta_k}\left(-\sum w_{jk} o_j + \theta_k\right)$$

$$= o_k{}^2 \frac{1 - o_k}{o_k}\left(\sum w_{jk} o_j + \theta_k\right)$$

$$= o_k(1 - o_k)\left(\sum w_{jk} o_j + \theta_k\right) \tag{4.24}$$

$$\frac{\partial E}{\partial c_k} = -(t_k - o_k)o_k(1 - o_k)\left(\sum w_{jk} o_j + \theta_k\right) \tag{4.25}$$

Therefore, the gain update expression for links connecting to output nodes is:

$$\Delta c_k(n+1)=\eta\,(t_k-o_k)o_k(1-o_k)\left(\sum w_{jk}o_j+\theta_k\right) \tag{4.26}$$

With momentum term, the gain update expression for links connecting output nodes is:

$$\Delta c_k(n+1)=\eta\,(t_k-o_k)o_k(1-o_k)\left(\sum w_{jk}o_j+\theta_k\right)+\mu\,\Delta c_k(n) \tag{4.27}$$

The calculation of $\frac{\partial E}{\partial c_j}$:

The calculation of $\frac{\partial E}{\partial c_j}$ is not very straightforward as the error depends on the output of the hidden as well as the output units. Therefore,

$$\frac{\partial E}{\partial c_j}=\frac{\partial\left[\frac{1}{2}\sum(t_k-o_k)^2\right]}{\partial c_j} \tag{4.28}$$

$$=\frac{\partial\left[\frac{1}{2}\sum(t_k-o_k)^2\right]}{\partial o_j}\frac{\partial o_j}{\partial c_j} \tag{4.29}$$

The subscripts j and k stand for hidden and output nodes respectively. To avoid confusion while using other subscripts, the superscripts h and o will be mentioned where necessary to stress the hidden and output nodes respectively.

Equation 4.29 is rewritten as

$$\frac{\partial E}{\partial c_j}=\frac{\partial\left[\frac{1}{2}\sum\left(t_k^o-o_k^o\right)^2\right]}{\partial o_j^h}\frac{\partial o_j^h}{\partial c_j}$$

$$=\frac{\partial\left[\frac{1}{2}\sum\left(t_k^o-o_k^o\right)^2\right]}{\partial o_k^o}\frac{\partial o_k^o}{\partial o_j^h}\frac{\partial o_j^h}{\partial c_j} \tag{4.30}$$

From Equation 4.23,

$$\frac{\partial o_k}{\partial o_j} = o_k{}^2 c_k w_{jk} \frac{1-o_k}{o_k}$$

$$= o_k(1-o_k)c_k w_{jk} \tag{4.31}$$

Now,

$$\frac{\partial\left[\frac{1}{2}\sum(t_k - o_k)^2\right]}{\partial o_k} = -(t_k - o_k) \tag{4.32}$$

$$\therefore \frac{\partial\left[\frac{1}{2}\sum_k(t_k - o_k)^2\right]}{\partial o_j} = -\sum_k c_k w_{jk} o_k(1-o_k)(t_k - o_k) \tag{4.33}$$

Now, let's calculate the term $\frac{\partial o_j}{\partial c_j}$

For the sigmoidal activation function,

$$o_j = \frac{1}{1+e^{-c_j\sum w_{ij}o_i + \theta_j}}$$

$$\frac{\partial o_j}{\partial c_j} = o_j(1-o_j)\left(\left(\sum_j w_{ij}o_i\right)+\theta_j\right) \tag{4.34}$$

Therefore, for each hidden unit *j*, substituting equations 4.28, 4.27 in Equation 4.22 we get,

$$\therefore \frac{\partial E}{\partial c_j} = \left[-\sum_k c_k w_{jk} o_k(1-o_k)(t_k - o_k)\right] o_j(1-o_j)\left(\left(\sum_j w_{ij}o_i\right)+\theta_j\right) \tag{4.35}$$

Therefore, the gain update expression for the links connecting hidden nodes is:

$$\Delta\ c_j(n+1) = \eta\left[\sum_k c_k w_{jk} o_k(1-o_k)\ (t_k - o_k)\right]\ o_j(1-o_j)\ \left(\left(\sum_j w_{ij}o_i\right)+\theta_j\right) \tag{4.36}$$

With momentum term, the gain update expression for links connecting hidden nodes is:

$$\Delta\ c_j(n+1) = \eta\left[\sum_k c_k w_{jk} o_k (1-o_k)\ (t_k - o_k)\right]\ o_j(1-o_j)\left(\left(\sum_j w_{ij} o_i\right) + \theta_j\right) + \mu\ \Delta\ c_j(n) \qquad (4.37)$$

4.6.2 Influence of using Adaptive Gain values with Sequential Mode of Training

The sequential mode of training requires an immediate updating of the weights, biases and gains after the presentation of a training example. An epoch is said to be complete after the presentation of the entire training set. A sum squared error value is calculated after the presentation of each training example and compared with the target error. Training is done on an epoch-by-epoch basis until the sum squared error value falls below the desired target error value.

The following iterative and coupled algorithm is proposed by the author for the sequential mode of training. Weights, biases and gains are calculated and updated for a training pair, which is being presented to the network. This algorithm is developed by the author and coded using C++ programming language. The algorithm uses the following terms.

```
For a given epoch,
    For each example,
        Calculate the weight and bias values using the previously converged gain
        value.                                                              (1)
        Use the weight and bias values calculated in step (1) to calculate the new
        gain value.                                                         (2)
        Repeat steps (1) and (2) by using the gain value calculated in step (2) in
        step (1) until the difference in consecutive weight, bias and gain values
        becomes less than the predefined value.
```

The speed of convergence achieved using this algorithm is demonstrated in the following example.

Consider a single input-output two-layer network with one hidden layer having three hidden nodes. The training data set is created by using the function $y = \sin(pi * x) \quad where\ x \in [0,1]$ and by adding an approximate 20% random Gaussian noise (Circles in Figure 4.17). The network is trained using 0.3 as the learning rate value to achieve a 6% target error, using the Gradient Descent training algorithm in a sequential mode with coupled and adaptive changes in weight, bias and gain values. The network is trained with an adaptive gain with a unit initial value of gain for all output as well as hidden nodes. The network took 199 epochs to learn the target function. During each epoch, for each training example, the gain, weight and bias values for all hidden and output nodes converged to achieve a root mean square error of 0.001 for gain, weight and bias values respectively. The network output (black continuous curve) is shown against the training data points (circles) in Figure 4.17.

The gain values for the three hidden nodes at the end of training are 3.99, 2.05 and 1.35 respectively.
The gain value for the output node at the end of training is 3.
The speed of convergence is high because the modified gain values are greater than unity.
The error versus number of epochs required to achieve the target error is plotted in Figure 4.18 (solid curve). The dotted curve represents the same graph for the network trained using a constant unit gain value in sequential mode.

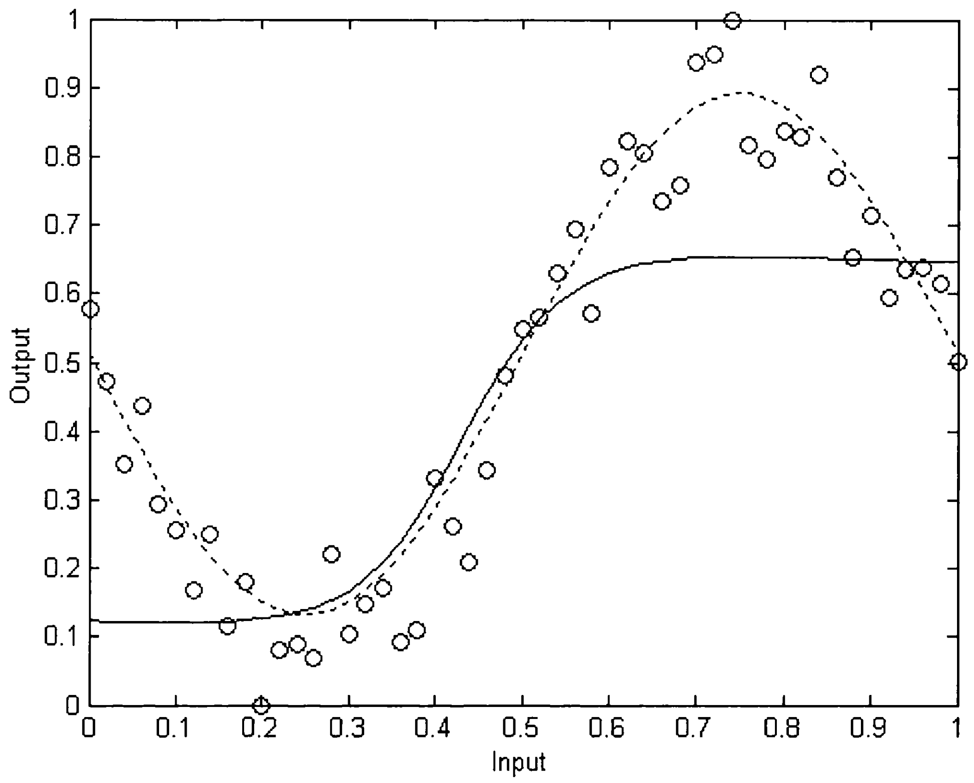


Figure 4.17 Output of neural network trained to learn a sine curve with 20% random Gaussian noise in sequential mode using the coupled algorithm.

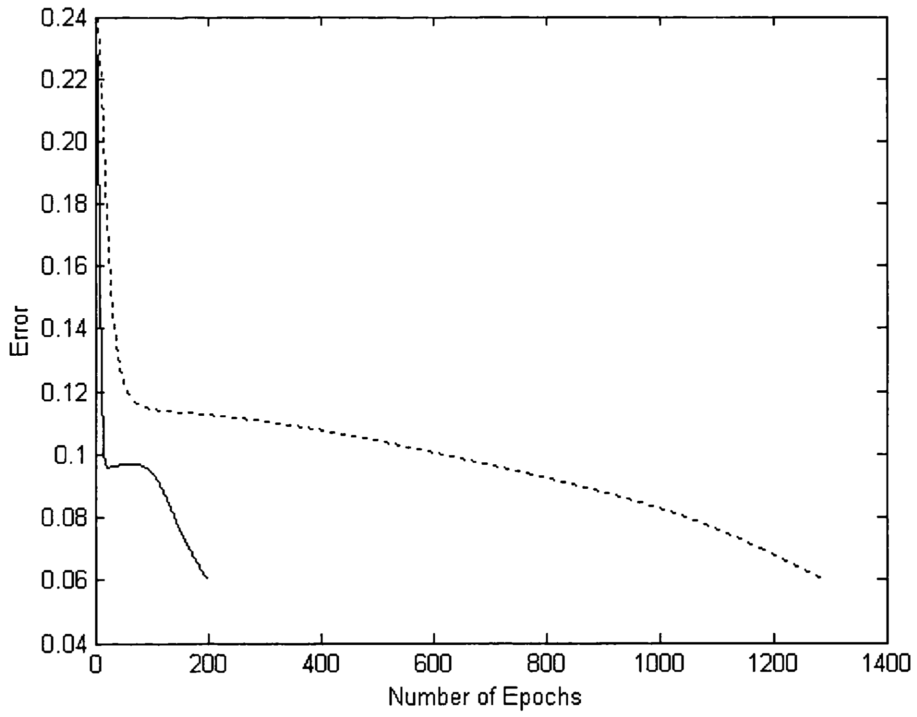


Figure 4.18 Error versus Number of Epochs required to achieve the Target Error of 0.06.

4.6.3 Influence of Adaptive Gain using Batch Mode of Training

As opposed to sequential training, in batch training mode, the weight, bias and gain values are updated after the presentation of an entire training set which constitutes an epoch.

The following iterative and coupled algorithm is proposed by the author for the batch mode of training. Weight, bias and gain values are calculated and updated after the presentation of all the training example pairs that constitute an epoch. This algorithm is developed by the author and coded using C++ programming language. The algorithm uses the following terms.

For a given epoch,

> Update the weight and bias values after the presentation of the entire example set using the previously converged gain value. (1)
>
> Use the weight and bias values calculated in step (1) to calculate the new gain value. (2)
>
> Repeat steps (1) and (2) by using the gain value calculated in step (2) in step (1) until the difference in consecutive weight, bias and gain values becomes less than a predefined value.

The speed of convergence achieved using this algorithm is demonstrated using the same example as illustrated in Section 4.6.1. The network took only one epoch to converge to around 6% error. During this epoch, hidden node 3 converged at the 45^{th} iteration, output node converged at the 147^{th} iteration, hidden node 1 and 2 both converged at the 699^{th} iteration to achieve a root mean square error of 0.001. The network output (black continuous curve) is shown against the training data points (circles) in Figure 4.19.

The values of gains for the three hidden nodes at the end of training are 5.18, 5 and 1.17 respectively.

The value of gain for the output node at the end of training is 0.69.

The speed of convergence is high because the modified gain values are greater than unity.

The error versus number of epochs required to achieve the target error is plotted in Figure 4.20 (solid curve). Figure 4.21 represents the same graph for the network trained using a constant unit gain value in batch mode.

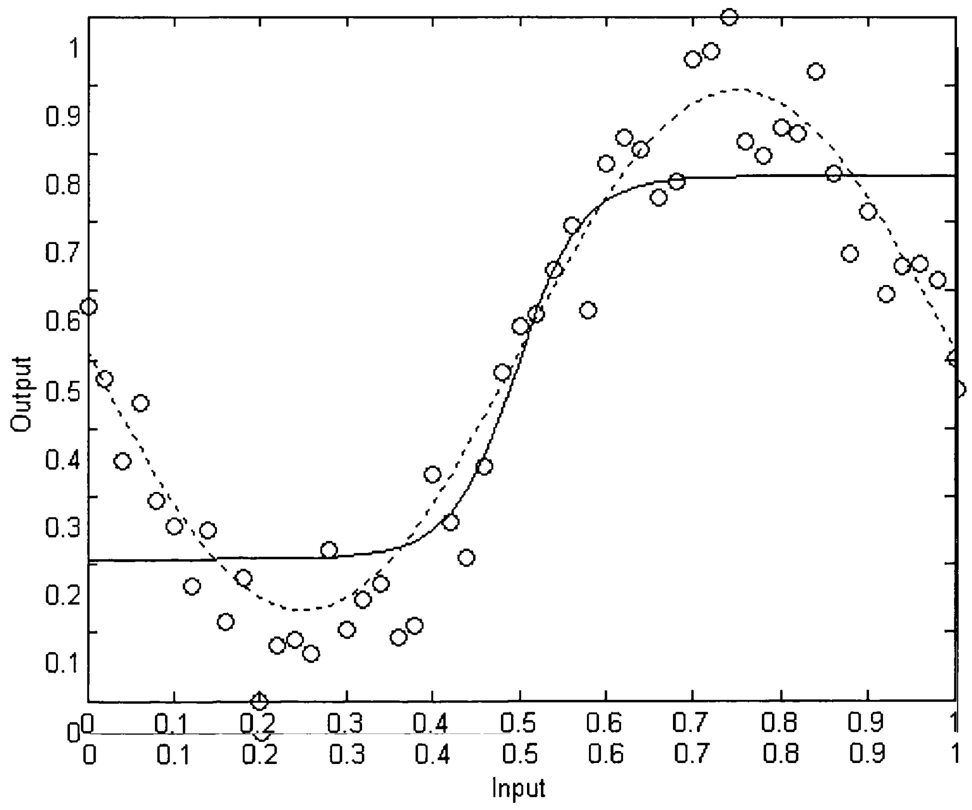


Figure 4.19 Output of neural network trained to learn a sine curve with 20% random Gaussian noise in batch mode using the coupled algorithm.

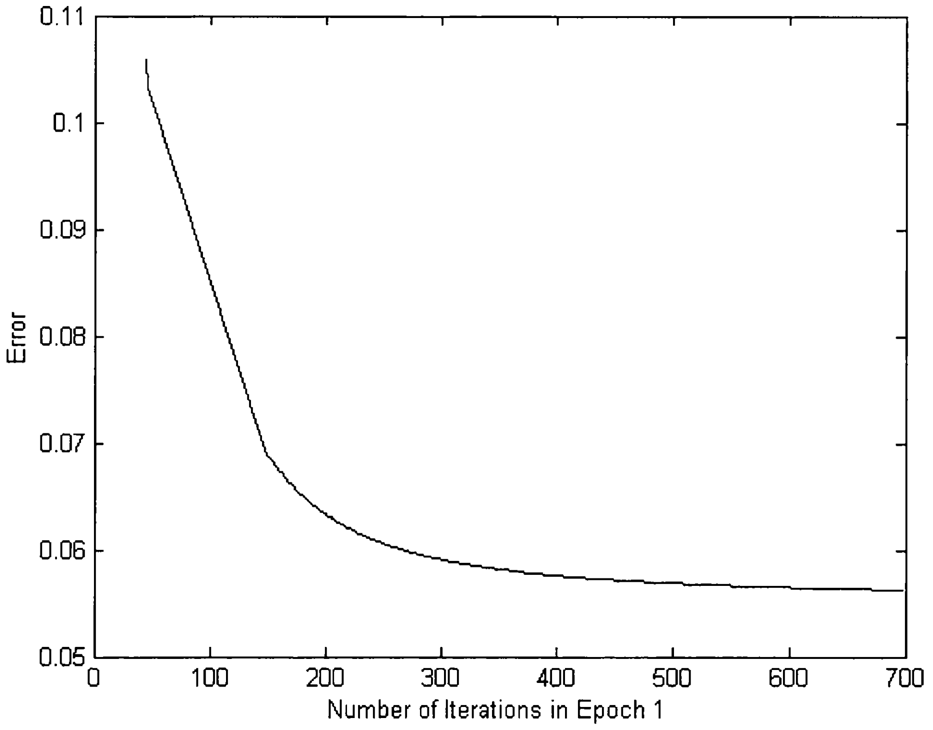


Figure 4.20 Error versus Number of Epochs required to achieve the Target Error of 0.06 using Adaptive Gain.

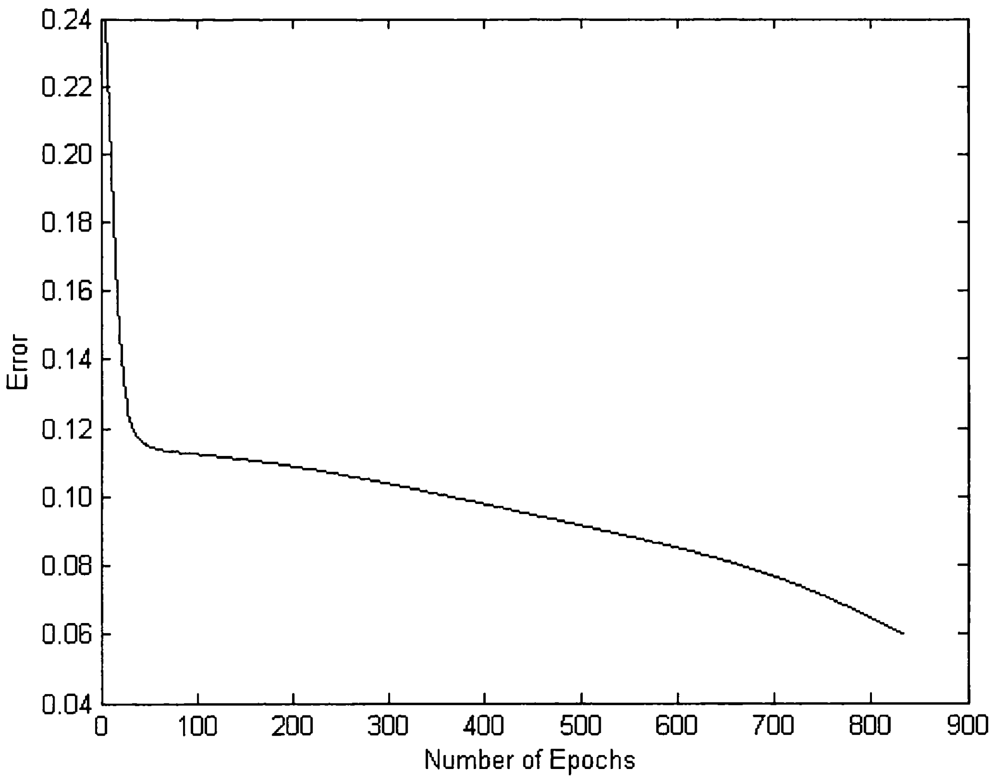


Figure 4.21 Error versus Number of Epochs required to achieve the Target Error of 0.06 using Constant Gain.

4.6.4 Benefits of Adaptive Gain

An advantage of using the adaptive gain procedure is that it is easy to introduce into a back-propagation algorithm, and it also accelerates the learning process without the need to invoke solution procedures other than Gradient Descent. Adaptive gain has a positive effect in the learning process by modifying the magnitude, and not the direction, of the weight change. This greatly increases the learning speed by amplifying the directions in the weight space that are successfully chosen by Gradient - Descent on weights [5]. A coupled algorithm has also been proposed in this thesis for the efficient calculation of the adaptive gain value in sequential as well as batch learning modes.

4.7 Conclusion

Most of the application oriented papers on neural networks tend to advocate that neural networks operate like a "magic black box", which can simulate the "learning from example" ability of our brain with the help of network parameters such as weights, biases, gain, hidden nodes etc. There are very few publications, or textbooks, which give a physical interpretation for various parameters used in the network. The influence of the variation of gain value on the performance of the network has been discussed in this chapter. It was observed that different values of gain used during the training process might not really achieve the desired positive control on all the output values. In fact, it is shown that the influence of variation in the gain value is similar to the influence of variation in the learning rate value. Furthermore, a relationship between the gain, the learning rate and the initial weights of a network has also been identified. Sample examples and calculations are shown, which support this relationship. This result has also been substantiated by independent research that was simultaneously done [1]. The results obtained from a case study using back propagation neural networks for modelling environmental systems, also indicate that learning rate, momentum and the gain of the transfer function have a significant impact on training speed, and not on generalisation ability. As a result, learning speed is affected by a combination of these factors, and the same learning speed can be achieved by using different combinations of these parameters [3]. However, higher values of learning rate and/or gain cause instability [4]. A coupled algorithm has been proposed in this thesis to change the gain value adaptively. The influence of adaptive gain on the performance of the network has also been analysed in this chapter using sequential as well as batch modes of training. Codes have been developed by the author using C++ programming language.

REFERENCES

[1] Thimm G., Moerland F., Fiesler E., "The Interchangeability of Learning Rate and Gain in Backpropagation Neural Networks", *Neural Computation*, vol **8**, no **2**, 1996, pp 451-460.

[2] Looney C.G., "Stabilization and Speedup of Convergence in Training Feed Forward Neural Networks", *Neurocomputing*, vol **10**, no **1**, 1996, pp 7-31.

[3] Holger R.M., Graeme C.D., "The Effect of Internal Parameters and Geometry on the Performance of Back-Propagation Neural Networks", *Environmental Modelling and Software,* vol **13**, no **1**, 1998, pp 193-209.

[4] Hollis P.W., Harper J.S., Paulos J.J., "The Effects of Precision Constraints in a Backpropagation Learning Network", *Neural Computation*, vol **2**, no **3**, 1990, pp 363-373.

[5] Kruschke J.K., Movellan J.R., "Benefits of Gain: Speeded learning and minimal hidden layers in bachpropagation networks", *IEEE Transactions on Systems, Man, and Cybernetics*, vol **21**, no **1**, 1991, pp 273-280.

[6] Han J, Moraga C, "The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning", *From Natural to Artificial Neural Computation Lecture Notes in Computer Science*, 930: 1995, pp195-201.

[7] Ransing R.S., "An Approach for the Causal Analysis in Casting Processes based on Probabilistic Analysis, Neural Network and Design Optimisation", *PhD*

Thesis, Institute of Numerical Methods in Engineering, Department of Civil Engineering, University of Wales Swansea, Chapter 4, March 1996.

APPENDIX

4A. Procedure for calculating the equation of a line to classify input pattern for a single layer network (Example 1):

The equation of a line for an output equal to 0.5 is derived as follows.

From Equation 4.1, the output of the j^{th} node is:

$$0.5 = \frac{1}{1+e^{-ca_{net,j}}}$$

where $a_{net,j}$ is the weighted sum of the inputs and c is the gain parameter.

$$\therefore 1+e^{-ca_{net,j}} = \frac{1}{0.5}$$

$$\therefore e^{-ca_{net,j}} = 2-1=1$$

$$\therefore -ca_{net,j} = 0$$

$$\therefore a_{net,j} = 0$$

$$\text{i.e. } x_1w_1 + x_2w_2 + b_1 = 0 \qquad (4.38)$$

where x_1 and x_2 are inputs and b_1 is the bias for the output node.

The equation of the line for an output value equal to 0.5 for any value of gain can be expressed as $x_1w_1 + x_2w_2 + b_1 = 0$.

The input patterns are linearly separable when the decision boundary, which partitions the input space into two regions, is a straight line (Figure 4.4). The decision rule for the classification is to assign a point (x_1, x_2) to region 1 if the net input $x_1w_1 + x_2w_2 + b_1 < 0$ and to region 2 if the net input $x_1w_1 + x_2w_2 + b_1 > 0$.

4B. Procedure for plotting of the curve that classifies the input pattern for a multi layer network (Example 2).

1. To find x_1 and x_2 we first have to calculate X_1 and X_2. X_1 and X_2 are outputs of the hidden nodes 1 and 2 respectively, which are also inputs to the output node. Therefore, the expression for the sum of weighted inputs to the output layer is:

$$X_1 w_5 + X_2 w_6 + b_3 = 0$$

$$\therefore X_2 = \frac{-b_3 - X_1 w_5}{w_6}$$

Using this we can find values of X_2 for different values of X_1. X_1 and X_2 are linearly separable. The decision rule for the classification is to assign a point (x_1, x_2) to region C if the net input $X_1 w_5 + X_2 w_6 + b_3 > 0$ and to region A if the net input $X_1 w_5 + X_2 w_6 + b_3 < 0$.

The second step is to derive an expression to find x_1 and x_2.

Output of the first hidden node $X_1 = \dfrac{1}{1 + e^{-c(x_1 w_1 + x_2 w_2 + b_1)}}$ (4.39)

Similarly, the output of the second hidden node $X_2 = \dfrac{1}{1 + e^{-c(x_1 w_3 + x_2 w_4 + b_2)}}$ (4.40)

From Equation (4.39) we have,

$$1 + e^{-c(x_1 w_1 + x_2 w_2 + b_1)} = \frac{1}{X_1}$$

$$e^{-c(x_1 w_1 + x_2 w_2 + b_1)} = \frac{1}{X_1} - 1$$

$$-c(x_1 w_1 + x_2 w_2 + b_1) = \ln\left(\frac{1}{X_1} - 1\right)$$

$$x_1 w_1 + x_2 w_2 = \frac{-\ln\left(\frac{1}{X_1} - 1\right)}{c} - b_1 \qquad (4.41)$$

Similarly from Equation (4.40) we have,

$$x_1w_3 + x_2w_4 = \frac{-\ln\left(\frac{1}{X_2} - 1\right)}{c} - b_2 \qquad (4.42)$$

Dividing Equation (4.41) by w$_1$ and Equation (4.42) by w$_3$, and then undertaking a subtraction results in:

$$\therefore x_2 = \frac{\frac{1}{w_1}\left[\frac{-\ln\left(\frac{1}{X_1} - 1\right)}{c} - b_1\right] - \frac{1}{w_3}\left[\frac{-\ln\left(\frac{1}{X_2} - 1\right)}{c} - b_2\right]}{\left(\frac{w_2}{w_1} - \frac{w_4}{w_3}\right)} \qquad (4.43)$$

Substituting values of x_2 in Equation (4.41) we get,

$$x_1 = \frac{\frac{-\ln\left(\frac{1}{X_1} - 1\right)}{c} - b_1 - x_2w_2}{w_1} \qquad (4.44)$$

Similarly we can find expressions for curves for various output values such as 0.1, 0.9 etc.

4C. Sample numerical calculations to illustrate the relationship between the Gain Value, Learning Rate Value and Initial Weight Values of the network.

Consider two multi layer networks as shown in Figures 4.13 and 4.14 with two input nodes with inputs $x_1 \in [0,1]$ and $x_2 \in [0,1]$ respectively, one hidden layer with two hidden nodes and one output node with output $a_j \in [0,1]$ each.

Sample calculations for Network 1 and Network 2 are as follows.

Network 1		Network 2	
weights	$w = [w_1, w_2, w_3, w_4, w_5, w_6]$	weights	$\overline{w} = cw$
bias	w_0	bias	$= cw_0$

Initial weights and biases for links connecting output node for both networks :

Initial weights = 0.996784, 0.517971

Initial bias = 0.735032

Initial weights and biases for links connecting the first hidden node for both networks:

Initial weights = -0.171131, -0.863562

Initial bias = -0.510387

Initial weights and biases for links connecting the second hidden node for both networks:

Initial weights = 0.499033, 0.00980157

Initial bias = 0.165547

gain	$c = 4$	gain	$\overline{c} = 1$
learning rate	$\eta = 0.25$	learning rate	$\overline{\eta} = c^2\eta = 0.3$

The network is trained for a target error equal to 0.01. The following sample calculations are shown for the first training example pair of the first epoch:

For first hidden node:

Sum of weighted inputs :

	$a_{net,j} = 0.242120$		$a_{net,j} = 0.968481$
Output :	$a_j = 0.724817$	Output :	$a_j = 0.724817$

For second hidden node:	
Sum of weighted inputs :	
$a_{net,j}$ = -0.240744	$a_{net,j}$ = -0.962975
Output : a_j = 0.276283	Output : a_j = 0.276283
For output node:	
Sum of weighted inputs	
$a_{net,j}$ = 132491	$a_{net,j}$ = 0.559963
Output : a_j = 0.629474	Output : a_j = 0.629475
Derivative = 0.932946	Derivative = 0.233236
Error Information term = -0.587265	Error Information term = -0.146816
For first hidden node:	
Derivative = 0.932946	Derivative = 0.233236
Error Information term = -0.587265	Error Information term = -0.146816
For second hidden node:	
Derivative = 0.799803	Derivative = 0.199951
Error Information term= -0.00115094	Error Information term= -
0.000287735	
For output node:	
Weight correction terms: -0.00798112,	Weight correction terms: -0.0319245,
-0.00204222	-0.0121689
Bias correction term: -0.0110112	Bias correction term: -0.0440449
For first hidden node:	
Weight correction terms: $-3.78981*10^{-5}$,	Weight correction terms: -0.000151592,
-0.000566903	-0.00226761
Bias correction term: -0.00109601	Bias correction term: -0.00438405
For second hidden node:	

Weight correction terms:	$-7.46201*10^{-7}$,	Weight correction terms:	$-2.98480*10^{-6}$
	$-1.11621*10^{-5}$		$-4.46485*10^{-5}$
Bias correction term:	$-2.15801*10^{-5}$	Bias correction term:	$-8.63205*10^{-5}$

The target error has been minimised in 1352 epochs and at the end of training we have the following results.

<u>Final weights and biases for links connecting first hidden node :</u>

Final weights	= -0.938798, 2.20565	Final weights	= -3.7551, 8.82252
Final bias	= -1.32148	Final bias	= -5.28591

<u>Final weights and biases for links connecting second hidden node :</u>

Final weights	= -2.55955, -1.42964	Final weights	= -10.238, -5.71878
Final bias	= 1.88456	Final bias	= 7.53814

<u>Final weights and biases for links connecting output node :</u>

Final weights	= 2.08198, -2.59675	Final weights	= 8.32778, -10.3871
Final bias	= 0.865063	Final bias	= 3.46011

Chapter 5

Neural Networks as a Regression Analysis Tool

In this chapter, the author discusses how neural network based models relate with regression analysis models, and also compares their advantages and limitations. Neural networks are used to model complex functional mappings when prior knowledge between input and output relationships is unknown. However, regression analysis may become advantageous if some knowledge of the input-output relationship is already known. As the objective of this research work is to explore various techniques so that prior knowledge can be embodied into a neural network like implementation, the similarities and dissimilarities between neural networks and regression analysis based models are explored.

5.1 Introduction

Neural networks have been used for a wide variety of applications where statistical models are traditionally employed. They have been used in classification problems such as categorising student applicants and determining the likelihood that they will enrol at an institution if offered a place [1] or predicting the academic success of MBA students [2]. There are many neural network applications in business, particularly in finance, e.g. bankruptcy prediction, stock market forecasting [3] and the prediction of exchange rate

[4] and many more [5]. Neural networks have also been used in applications such as modelling the fluidised bed granulation process [6], prediction of Ozone indices [7], spatial prediction of fire ignition probabilities [8]. These problems would normally be solved through classical statistical models such as *discriminant analysis* [9] and *multiple regression analysis* [10-13]. Warner and Misra [14] have cited a few more examples in which comparative studies between statistical methods and neural networks have been done.

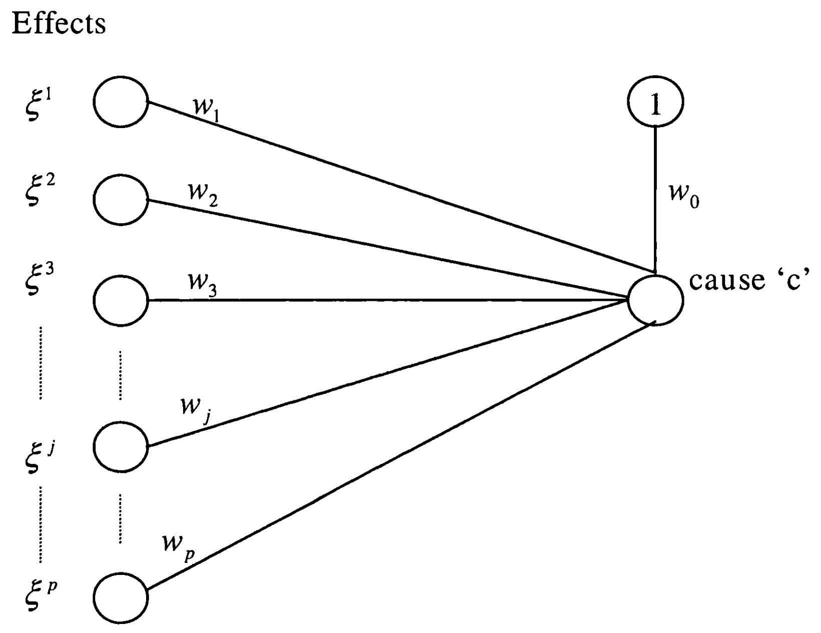


Figure 5.1: Schematic representation of a 'cause-effect relationship'.

If a linear multivariate regression analysis [15] is used for learning cause and effect relationships, as proposed in this research work, the belief values representing the strength of the effects will be associated with a belief value which quantifies the extent of occurrence of cause using an expression similar to Equation 5.1.

$$\text{Belief in cause '}c\text{'} = w_0 + w_1\xi^1 + w_2\xi^2 + w_3\xi^3 + \ldots + w_j\xi^j + \ldots + w_p\xi^p \tag{5.1}$$

In this equation the belief values for the 'p' effects, associated with a cause 'c', are represented by 'p' variables $\xi^1, \xi^2, \xi^3, \ldots \xi^j, \ldots \xi^p$ (as shown in Figure 5.1). Variables $w_j \; (j = 0 \; to \; p)$ are referred to as either regression coefficients (in a regression analysis context) or weights (in a neural network context). These coefficients, or weights are generally considered as independent variables and are mostly determined using least square minimisation techniques by comparing the belief value in the cause, calculated by Equation 5.1, with a previously known value for the same set of input values.

For a non-linear regression analysis, ξ^j $(j = 1\, to\, p)$ (in Equation 5.1) are replaced by '*m*' different functions z_i $(i = 1\, to\, m)$ ranging from simple linear polynomials to higher order, non-linear polynomials, logarithmic or exponential functions. Multi-layered feed forward neural network techniques [9], and a range of methods proposed in the family of intrinsically linear, multivariate regression analysis [15], generalise Equation 5.1 in the following way:

$$\text{Belief in cause '}c\text{'} = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + \ldots + w_i z_i + \ldots + w_m z_m \qquad (5.2)$$

where each z_i $(i = 1\, to\, m)$ represents a function of ξ^j $(j = 1\, to\, p)$

In this chapter, neural network techniques are compared with regression analysis methods in order to develop a method, which will retain the advantages of both techniques and at the same time overcome the limitations.

5.2 Similarity of Neural Network Algorithm with Generalised Regression Analysis

An example of a multi-layered network is shown in Figure 5.2. In this network there are '*l*' input nodes, '*m*' hidden nodes and '*n*' output nodes. The feed forward neural network can have any number of hidden layers with a variable number of hidden units per layer. If the neural network algorithms are expressed as a regression tool, the expression for function 'z_i' corresponding to Equation 5.2 and Figure 5.2 is obtained as follows.

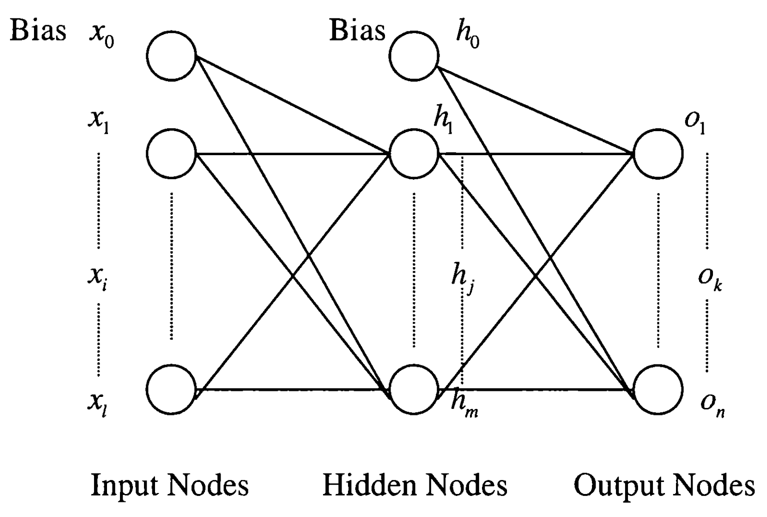


Figure 5.2: An example of a feed-forward network having two layers of adaptive weights. Biases are represented as weights associated with an extra input node x_0 and an extra hidden node h_0.

The output of the j^{th} hidden node is obtained by a linear combination of l input values, which by adding a bias term, gives an expression for the net input to the j^{th} hidden node as

$$a_{net,j} = \sum_{i=1}^{l} w_{ji} x_i + w_{j0} \tag{5.3}$$

We rewrite Equation 5.3 by including the bias term as a weight value connected to an input node of unit value.

$$a_{net,j} = \sum_{i=0}^{l} w_{ji} x_i \tag{5.4}$$

The output of the hidden node 'j' is then obtained by transforming the net input value, as given in Equation 5.4, and then using a non-linear activation function $g(.)$ to give

$$\begin{aligned} h_j &= g\left(a_{net,j}\right) \\ &= z_j\left(x_i\right); \quad \left(i = 0\ to\ l\right) and \left(j = 0\ to\ m\right) \end{aligned} \tag{5.5}$$

The output of the output layered nodes is obtained by transforming the output value of hidden nodes using a second layer of processing elements. Thus, for each output node *k*, we calculate a linear combination of the output values of hidden nodes as follows.

$$a_{net,k} = \sum_{j=0}^{m} w_{kj} h_j \tag{5.6}$$

An output value of the k^{th} output node is then obtained by transforming this linear combination, using a linear activation function, to give

$$o_k(\mathbf{x}) = a_{net,k} \tag{5.7}$$

Combining Equations 5.4, 5.5, 5.6 and 5.7, we obtain an explicit expression to relate the output and input variables which represent the network as shown in Figure 5.2:

$$\begin{aligned} o_k(\mathbf{x}) &= \sum_{j=0}^{m} w_{kj} g\left(\sum_{i=0}^{l} w_{ji} x_i\right) \\ &= \sum_{j=0}^{m} w_{kj} z_j(\mathbf{x}) \end{aligned} \tag{5.8}$$

Thus, the regression analysis expression, as given in Equation 5.2, will represent a forward pass in the neural network algorithm if the functions 'z_j' are given by Equation 5.5. The network thus represents a multivariate non-linear functional mapping.

5.3 Neural Networks and Regression Analysis

The advantages of using neural networks and regression analysis, for associating belief values in causes with belief values in effects, are given below.

1. Research on the approximation capabilities and separability of multi layer feed forward neural networks has been done by e.g. Hornik [16], Hornik, Stinchcombe, White [17-18], Funahashi [19], Huang and Babri [20], Leshno, Vladimir, Pinkus,

Schocken [21], Gallant and White [22], Bahrami [23] and Koiran [24]. Their studies show that multi layer feed forward neural networks, with a single hidden layer with a sigmoidal activation function for hidden layered nodes, are a class of universal approximators, that are capable of approximating an arbitrary function and its derivatives to any desired degree of accuracy, provided a sufficient number of hidden units are available. Furthermore, Huang, Chen and Babri [25] showed that these networks could form disjoint decision regions with arbitrary shapes in multidimensional cases.

2. The neural network model does not assume any functional form for the relationship between the independent and dependent variables.

3. Regression analysis performs better when some form of an underlying relationship between the dependent and independent variables is known.

4. Regression analysis can be a better alternative when the training data set is small.

5. If no data is available beyond the range of required predictions, then extrapolation abilities of regression analysis are better as compared to neural networks.

The next two subsections briefly explain the limitations of both approaches.

5.3.1 Limitations of Neural Networks for Learning Cause and Effect Relationships

1. It is difficult to select optimal values for certain network parameters e.g. the number of hidden nodes, the value for the learning rate parameter η and the initial weight values. It is also not easy to decide when to stop the training process so as to prevent the network from over fitting the data. The process of determining appropriate values for these variables is often of a 'trial and error' nature, and therefore, it can be time consuming. Hence it is difficult to use this technique, as a robust tool, by

end users in a foundry environment or any other manufacturing industry, to analyse cause and effect relationships.

2. Conventional neural network techniques *cannot* constrain the shape of the resulting input-output mapping function to previously known patterns. It is also quite likely that the shape of the resulting input-output mapping function is unrealistic, if the training data is noisy and incomplete.

3. There is also a risk of the network over-fitting the training data. The noise and duplication in the data will further complicate the issue as it may result in the loss of the generalisation ability of the network.

4. Extrapolation abilities of neural networks are poor.

These limitations are also true for semantically constrained neural network architecture as proposed in a previous work [26].

5.3.2 Limitations of Regression Analysis for Learning Cause and Effect Relationships

1. To decide the order of polynomials used in the regression model, some knowledge of the input and output analysis is necessary.

2. We can approximate a function to a reasonable accuracy, provided there are a sufficiently large numbers of higher ordered terms in the polynomial. However, this results in an exponential increase in the number of unknowns. (For d input variables and one output variable, for an M-th order polynomial, the number of independent variables grow as d^M). This renders the regression analysis technique as an impractical tool for analysing cause and effect relationships.

5.4 Conclusion

Codes for traditional neural networks and regression analysis techniques have been developed by the author and written in Matlab. Neural network techniques and regression analysis methods have been discussed briefly and comments have been made on their advantages and limitations. Regression analysis imposes a functional form on the data as a result of which it has better extrapolation abilities as compared to neural network techniques. On the contrary, neural networks extract the functional form from the data and hence, are useful when we do not know the underlying relationship between the independent and dependent variables. Also, the network architecture for neural networks is not unique, as the number of hidden units is decided by trial and error, which is generally time consuming. For regression analysis, the number of unknowns increases exponentially with the dimensionality of the input space and this is a serious limitation. As a result of this, large training data becomes necessary. For analysing cause and effect relationships, it is generally difficult to get good quality training data. In the next Chapter, a novel method has been proposed which appears to retain the advantages of both techniques and overcomes their limitations.

REFERENCES

[1] Walczak S., Sincich T., "A comparitive analysis of regression and neural networks for university admissions", *Information Sciences*, vol 119,no 1-2, pp 1-20, Oct 1999.

[2] Wilson R.L., Hardgrove B.C., "Predicting Graduate Student Success in an MBA Program – Regression Versus Classification", *Educational and Psychological Measurement*, vol 55, no 2, pp 186-195, Apr 1995.

[3] Fadlalla A., Chien-Hua L., "An Analysis of the Applications of Neural Networks in Finance", *Interfaces*, vol 31, no 4, pp 112-122, July-August 2001.

[4] Hann T.H., Steurer E., "Much ado about nothing? Exchange Rate Forecasting: Neural Networks vs. Linear Models using monthly and weekly data", *Neurocomputing*, vol 10, pp 323-339, 1996.

[5] Vellido A., Lisboa P.J.G., Vaughan J., "Neural Networks in business: a Survey of applications (1992-1998)", *Expert Systems with Applications*, vol 17, pp 51-70, 1999.

[6] Murtoniemi E., Yliruusi J., Kinnunen P., Merkku P., Leiviska K., "The Advantages by the use of Neural Networks in Modelling the Fluidized –Bed Granulation Process", *International Journal of Pharmaceutics*, vol 108, no 2, pp 155-164, Aug 1 1994.

[7] Soja G., Soja A.M., "Ozone Indices based on simple meteorological parameters: potentials and limitations of regression and neural network models", *Atmospheric Environment*, vol 33, no 26, pp 4299-4307, Nov 1999.

[8] de Vasconcelos M.J.P., Silva S., Tome M., Alvim M., Pereira J.M.C., "Spatial Prediction of Fire Ignition Probabilities: Comapring Logistic Regression and Neural Networks", *Photogrammetric Engineering and Remote Sensing*, vol 67, no 1, pp 73-81, Jan 2001.

[9] Bishop C.M., "Neural Networks for Pattern Recognition", Oxford University Press, 1995 (ISBN:0-19-853864-2).

[10] Lapin L.L., "Statistics For Modern Business Decisions", Third Edition, (ISBN: 0-15-583743-5).

[11] Reed R.D., Marks R.J., "Neural Smithing - Supervised Learning in Feedforward Artificial Neural Networks", The MIT Press, 1999 (ISBN: 0-262-18190-8).

[12] Principe J.C., Euliano N.R., Lefebvre W.C., "Neural and Adaptive Systems – Fundamentals Through Simulatios", John Wiley and Sons, Inc., 1999 (ISBN: 0-471-35167-9).

[13] Kleinbaum D.G., Kupper L.L., Muller K.E., Nizam A., "Applied Regression Analysis and Other Multivariable Methods", Duxbury Press, 1998 (ISBN: 0-534-20910-6).

[14] Warner B., Misra M., "Understanding Neural Networks as Statistical Tools", *The American Statistician*, vol. 50(4), pp 284-293, November 1996.

[15] Freund R.J. and Wilson W.J., "Regression Analysis: Statistical Modelling of a Response Variable", Academic Press, 1998 (ISBN: 0-12-267475-8).

[16] Hornik K., "Approximation Capabilities of Multilayer Feedforward Networks", *Neural Networks*, vol 4, no 2, pp 251-257, 1991.

[17] Hornik K., Stinchcombe M., White H., "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, vol 2, pp 359-366, 1989.

[18] Hornik K., Stinchcombe M., White H., "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks", *Neural Networks*, vol 3, no 5, pp 551-560, 1990.

[19] Funahashi K., "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, vol 2, pp 183-192, 1989.

[20] Huang G.B., Babri H.A., "Upper Bounds on the Number of Hidden Neurons in Feedforward Networks with Arbitrary Bounded Nonlinear Activation Functions", *IEEE Transactions on Neural Netowrks*, vol 9, no 1, pp 224-229, January 1998.

[21] Leshno M., Vladimir Y.L., Pinkus A., Schocken S., "Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function", *Neural Networks*, vol 6, no 6, pp 861-867, 1993.

[22] Gallant A.R. and White H., "On learning the Derivatives of an Unknown Mapping With Multilayer Feedforward Networks", *Neural Networks*, vol 5, no 1, pp 129-138, 1992.

[23] Bahrami M., "Issues on Representational Capabilities of Artificial Neural Netowrks and Their Implementation", *International Journal of Intelligent Systems*, vol 10, no 6, pp 571-579, 1995.

[24] Koiran P., "On the Complexity of Approximating Mappings Using Feedforward Networks", *Neural Networks*, vol 6, no 5, pp 649-653, 1993.

[25] Huang G.B., Chen Y.Q., Babri H.A., "Classification Ability of Single Hidden Layer Feedforward Neural Networks", *IEEE Transactions on Neural Networks*, vol 11, no 3, pp 799-801, May 2000.

[26] Ransing R. S., "An approach for the Causal Analysis in Casting Processes based on the Probabilistic Annalysis, Neural Network and Design Optimisation", *PhD Thesis*, Inst. of Numerical Methods in Engg., University of Wales Swansea, Swansea, UK, 1996.

CHAPTER 6

Lagrange Interpolation Polynomial Regression Analysis for studying Causal Relationships

This chapter proposes a method for determining the likelihood of a cause of one or more effects, in which training data relating to previously identified relationships between one or more causes and one or more effects is used to learn the cause and effect relationship. A number of reference points are chosen in the input space created by belief values representing the strength of the effects. A Lagrange Interpolation polynomial and a weight value is associated with each of the said reference points. To reduce the number of independent unknowns in the network, the reference points are divided into two categories, referred to as primary and secondary reference points. Weight values associated with these primary reference points are considered as independent variables (primary weight values) and other weight values, which are associated with secondary reference points (secondary weight values), linearly depend on one or more primary weight values. The belief value in the occurrence of the likely causes of one or more given effects is determined using this method.

6.1 Introduction

As described in earlier chapters, one of the major limitations of using neural network techniques is that any known information about the cause and effect relationship cannot effectively be stored within the network. As a result, the network performance becomes poor when the training data is noisy and may result in possible contradicting values. Also, in many real situations very few good quality training examples are generally available. As a result, to achieve the aims of this research it is necessary to a-priori store any known information about the cause – effect relationship within the network. This may change the neural network architecture, however, the "learning from examples" ability of the network should be unaltered.

As discussed in Chapter 2, the semantically constrained neural network [6] incorporates some extra information about causal connectivity into the network and then modifies the learning algorithm accordingly. However, this is certainly not sufficient. From the authors' discussion with foundry men as well as medical doctors, it was realised that the belief variation in the occurrence of a cause, with respect to a change in the belief value of the occurrence of an effect, follows a pattern. Such a variation is generally linear, quadratic or cubic and certainly not an arbitrary higher ordered polynomial. The network can achieve good extrapolation abilities if the computational algorithm uses this information along with the "learning from examples" ability of neural networks to calculate a belief value, which quantifies the extent of occurrence of a cause, given belief values, which quantify the occurrence, or non-occurrence, of associated effects of the cause. Some of the examples of the effects of a cause are the "symptoms" shown by patients in the medical domain, "defects" occurring in components in manufacturing industry or "effects" as generally meant in any "cause and effect" diagram.

A few examples of the variation in belief values in the occurrence of a cause, given a variation in the symptom strength, are given below. Medical experts generally characterise the strength of a symptom by adjectives e.g. *low*, *medium*, *high* and *very high* fever. Similarly the belief in the proposition that 'typhoid is a cause for a symptom fever' may also be characterised as *low*, *medium*, *high* and *very high*. The belief variation in such a one-dimensional cause and effect relationships can be graphically plotted. If the chest pain is *very high* then the belief that the patient is suffering from a 'heart attack' is *high* and the belief that the patient has a 'hair fracture or small muscular twist in the chest' is *low*. Examples have been taken from the medical field, as they are easier to visualise. Such knowledge needs to be created for all symptoms and causes. One-dimensional cause and effect relationships are presented below, as these relationships become more complicated in a two, three or higher dimensional input space.

The 'Output Value' in the following Figures (6.2a-6.2e) represents a belief value in the occurrence of the cause corresponding to the strength of the effect. These Figures (6.2a-6.2e) show a graphical representation of some general one-dimensional cause and effect relationships (Figure 6.1).

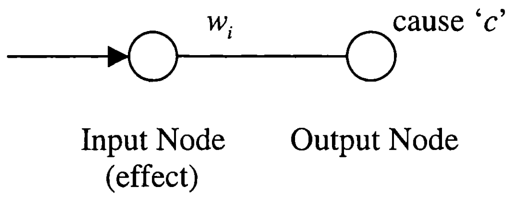
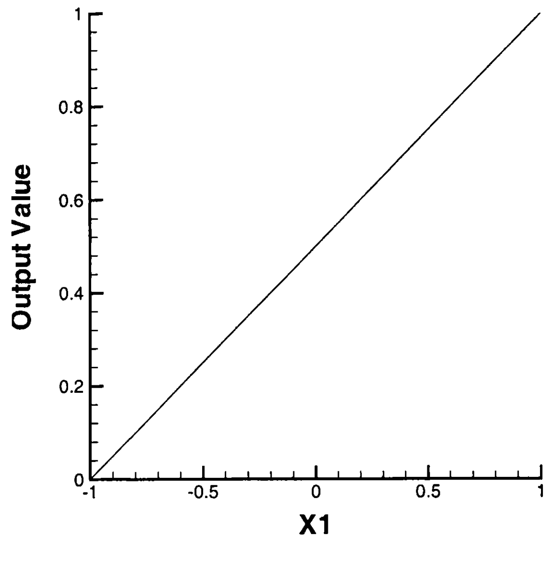


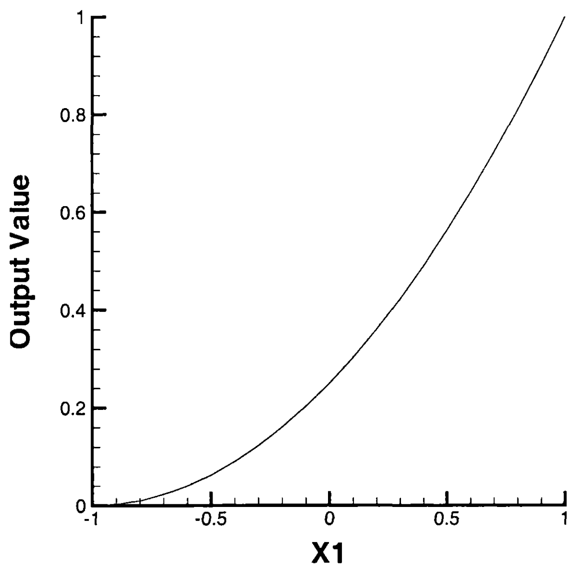
Figure 6.1: Schematic representation of a 'single effect - cause relationship'.



<u>X1: The strength of a symptom-'Fever'.</u>

Output Value: Belief that 'Typhoid' is a cause for a symptom- 'Fever'.

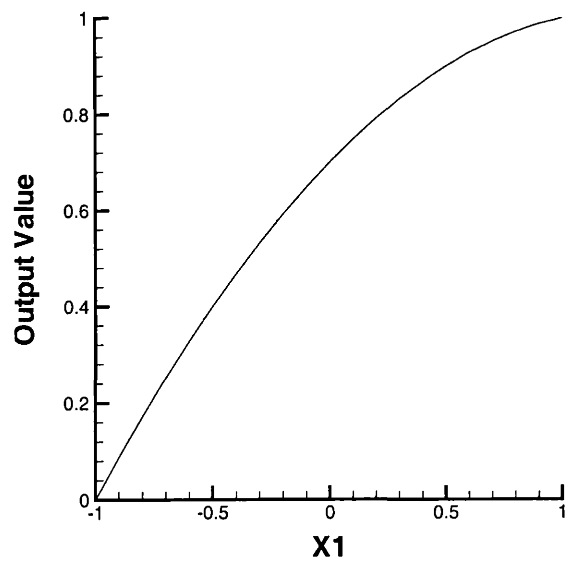
Figure 6.2a



<u>X1: The strength of a symptom-‘Chest Pain’.</u>

Output Value: Belief that ‘Heart Attack’ is a cause for a symptom- ‘Chest Pain’.

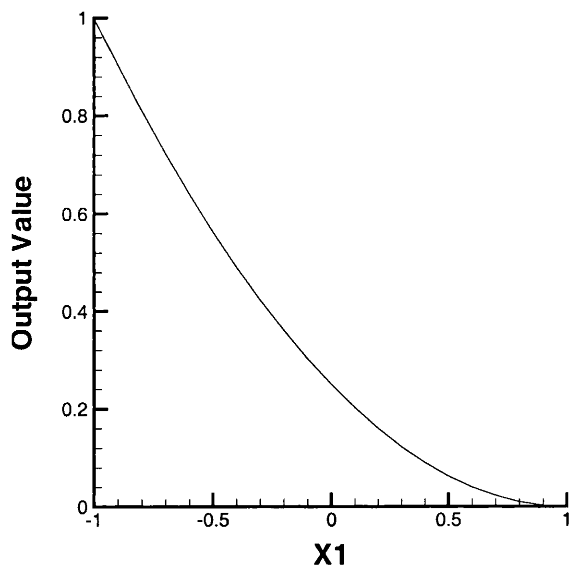
Figure 6.2b



<u>X1: The strength of a symptom- ‘Fever’.</u>

Output Value: Belief that ‘Brain Fever (Encephalitis)’ is a cause for a symptom- ‘Fever’.

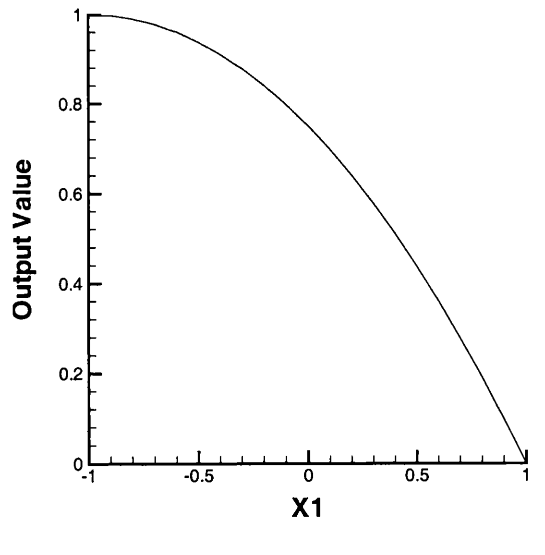
Figure 6.2c



<u>X1: The strength of a symptom-'Chest Pain'.</u>

Output Value: Belief that 'Hair Fracture in Ribs or Muscular Twist' is a cause for a symptom- 'Chest Pain'.

Figure 6.2d



<u>X1: The strength of a symptom-'Fever'.</u>

Output Value: Belief that 'Over-Exertion' is a cause for a symptom- 'Fever'.

Figure 6.2e

Figures 6.2a to 6.2e are graphical representations of some general one-dimensional cause and effect relationships with associated belief values (based on the network illustrated in Figure 1). Figures 6.2a to 6.2e show possible examples of the variation in belief values, representing the extent of occurrence of the cause (output value), with respect to the belief values, representing the strength of one of the associated effects.

A linear variation in belief values is shown in Figure 6.2a, in which when the belief value representing the strength of the symptom 'Fever' is at its minimum, the belief that 'Typhoid' is a cause for a symptom- 'Fever' is also at its minimum. As the strength of symptom or effect increases, the belief value in the occurrence of the cause also linearly increases.

A quadratic variation of cause and effect is shown in Figure 6.2b, in which when the belief value representing the strength of the effect or symptom 'Chest Pain' is at its minimum, then the belief that a 'Heart Attack' is a cause for a symptom- 'Chest Pain' is also at its minimum. As the strength of the effect starts to increase, the belief value in the occurrence of the corresponding cause also starts to slowly increase. As the strength of the effect increases to about half of its maximum value, so the belief value in the occurrence of the cause suddenly increases and reaches its maximum value when the strength of the effect reaches its maximum value.

A quadratic variation of cause and effect is shown in Figure 6.2c, in which, when the belief value representing the strength of the effect or symptom 'Fever' is at its minimum, then the belief that 'Brain Fever (Encephalitis)' is a cause for a symptom- 'Fever' is also at its minimum. As the strength of the effect starts to increase, the belief value in the occurrence of the cause quickly starts to increase. When the strength of the effect is around half of its maximum value, the rate of increase in the belief value of the occurrence of the related cause slows down and reaches its maximum value when the strength of the effect also reaches its maximum value.

A quadratic variation of cause and effect is shown in Figure 6.2d, in which when the belief value representing the strength of the effect or symptom 'Chest Pain' is at its minimum, then the belief that 'Hair Fracture in Ribs or Muscular Twist' is a cause for a symptom- 'Chest Pain' is at its maximum. As the strength of the effect starts to increase, there is a quick reduction in the belief value in the occurrence of the cause. As the strength of the effect increases to about half of its maximum value, the belief value in the occurrence of the cause slowly decreases and reaches its minimum value when the strength of the effect is at its maximum.

A quadratic variation of cause and effect is shown in Figure 6.2e, in which when the belief value representing the strength of the effect or symptom 'Fever' is at its minimum, then the belief that 'Over-Exertion' is a cause for a symptom- 'Fever' is at its maximum. As the strength of the effect starts increasing, the belief value in the occurrence of the cause slowly starts to decrease. When the strength of the effect reaches around half of its maximum value, the belief value in the occurrence of the cause starts to decrease quickly and reaches its minimum when the strength of the effect is at its maximum.

The belief value quantifying the occurrence, or non-occurrence, of an effect associated with a particular cause is generally normalised between +1 to –1 or 0 to 1 respectively. This belief value is also interpreted as being the strength of the effect. The belief value, which quantifies the extent of occurrence of the cause under consideration, is also normalised from zero to unity or –1 to +1 to represent non-occurrence and occurrence, respectively.

The quadratic and linear relationships given in Figure 6.2a to 6.2e can be modelled by corresponding quadratic or linear polynomials based on regression analysis techniques. However, as described in Chapter 5, multivariate non-linear regression analyses are not suitable for analysing cause and effect relationships, as the regression analysis will result in an excessively large number of unknown parameters.

Also, if more than one effect is associated with a cause, the causes shown in Figures 6.2a to 6.2e will generalise to multi-dimensional hyper surfaces. It is assumed that these hyper surfaces are also smoothed in a similar manner to their one-dimensional counterparts. Thus, the new computational procedure should be able to generate such smooth hyper surfaces by using the corresponding one-dimensional curves. For the first time, the use of Lagrange Interpolation Polynomials has been proposed to construct the multi dimensional hyper surfaces from linear, quadratic or cubic one-dimensional Lagrange Interpolation Polynomials.

6.2 Construction of Multi Dimensional Lagrange Interpolation Polynomials

The proposed method calculates a belief value ξ, which quantifies the extent of occurrence of a cause, given belief values which quantify the occurrence or non-occurrence of associated effects of the cause. This belief value is also interpreted to represent the strength of the effect, which is normalised between +1 to -1. The belief value, which quantifies the extent of occurrence of the cause under consideration is also normalised from zero to unity, to represent non-occurrence and occurrence of the said cause respectively.

The relationship between the belief value, representing the strength of the effect, and the belief value, representing the extent of occurrence of the cause, is assumed to be either linear, quadratic, cubic and so on (Figures 6.2a to 6.2e). The order of the relationship (e.g. one for linear, two for quadratic, three for cubic etc.) can either be given or calculated iteratively starting from one. To define an n^{th} order relationship along one-dimension, then $(n+1)$ reference points which are equidistant between –1 to +1, are chosen. For each reference point 'i' (i = 1 to n + 1), a one-dimensional Lagrange Interpolation Polynomial is constructed based on the following formula:

$$\begin{aligned} l_i(\xi) &= l_k^n(\xi) \\ &= \frac{\xi-\xi_0}{\xi_k-\xi_0} * \frac{\xi-\xi_1}{\xi_k-\xi_1} * \frac{\xi-\xi_2}{\xi_k-\xi_2} * \dots * \frac{\xi-\xi_{k-1}}{\xi_k-\xi_{k-1}} * \frac{\xi-\xi_{k+1}}{\xi_k-\xi_{k+1}} * \dots * \frac{\xi-\xi_n}{\xi_k-\xi_n} \end{aligned} \tag{6.1}$$

where,

n: order of the Lagrange Interpolation Polynomial (one for linear, two for quadratic, etc.)

k: A reference point at which the one-dimensional Lagrange Interpolation

Polynomial $l_k^n(\xi)$ is constructed. (k ranges from 0 to n).

i: Ranges from one to total number of reference points i.e. $(n+1)$.

$\xi_0, \xi_1, \xi_2 \ldots \xi_n$ are $(n+1)$ equidistant reference points from $\xi_0 = -1$ to $\xi_n = +1$ with corresponding $(n+1)$ Lagrange Interpolation Polynomials as given by Equation (6.1).

The variable ξ, which stores the belief value representing the strength of the corresponding effect, ranges from –1 to +1. For a "single effect – cause relationship", the Lagrange Interpolation Polynomial is one-dimensional and the reference points are drawn along this dimension. If the number of associated effects for a given cause is 'p', the Lagrange Interpolation Polynomial at a reference point 'i' will be 'p' dimensional and is given by the following equation:

$$l_i\left(\xi^1, \xi^2, \xi^3, \ldots \xi^j, \ldots \xi^p\right) = l_{k_1}^{n_1}\left(\xi^1\right) * l_{k_2}^{n_2}\left(\xi^2\right) * \ldots * l_{k_j}^{n_j}\left(\xi^j\right) * \ldots * l_{k_p}^{n_p}\left(\xi^p\right) \tag{6.2}$$

where,

$$l_{k_j}^{n_j}\left(\xi^j\right) = \frac{\xi^j - \xi_0^j}{\xi_{k_j}^j - \xi_0^j} * \frac{\xi^j - \xi_1^j}{\xi_{k_j}^j - \xi_1^j} * \frac{\xi^j - \xi_2^j}{\xi_{k_j}^j - \xi_2^j} * \ldots * \frac{\xi^j - \xi_{k_j-1}^j}{\xi_{k_j}^j - \xi_{k_j-1}^j} * \frac{\xi^j - \xi_{k_j+1}^j}{\xi_{k_j}^j - \xi_{k_j+1}^j} * \ldots * \frac{\xi^j - \xi_{n_j}^j}{\xi_{k_j}^j - \xi_{n_j}^j} \tag{6.3}$$

n_j: The order of one dimensional Lagrange Interpolation Polynomial $\left(l_{k_j}^{n_j}\left(\xi^j\right)\right)$ corresponding to j^{th} dimension that represents the relationship between j^{th} effect and the cause under consideration.

k_j: Reference point along j^{th} dimension, at which the one dimensional Lagrange Interpolation Polynomial $l_{k_j}^{n_j}\left(\xi^j\right)$ is evaluated. (k_j ranges from 0 to n_j.)

$\xi_0^j, \xi_1^j, \xi_2^j, \ldots, \xi_{n_j}^j$ are (n_j+1) reference points along the j^{th} dimension.

i: For a 'p' dimensional case, 'i' ranges from one to total number of reference points 'q (as given by Equation 6.4)'.

As k_j independently varies from 0 to n_j for each Lagrange Interpolation Polynomial,

$$q = (n_1+1)*(n_2+1)*(n_3+1)*\ldots*(n_j+1)*\ldots*(n_p+1) \tag{6.4}$$

It should be noted that, the co-ordinates for a reference point 'i', corresponding to each dimension, are given as $(k_1, k_2, \ldots, k_j, \ldots, k_p)$.

6.3 Construction of the Decision Hyper Surface using Lagrange Interpolation Polynomials

The Lagrange Interpolation Polynomials given by Equation (6.2) are used to generate the decision surface representing belief values in the occurrence of a cause using the Equation 6.5.

The belief value in the occurrence of a cause, based on the known belief values $\left(\xi^j,\ j = 1\ to\ p\right)$ quantifying the strength of 'p' associated effects, is given by the following equation:

$$\text{The belief value in the cause} = \sum_{i=1}^{q} w_i l_i\left(\xi^1, \xi^2, \ldots, \xi^p\right) \qquad (6.5)$$

where,

q: Total number of reference points.

$l_i\left(\xi^1, \xi^2, \ldots, \xi^p\right)$ is given by Equation (6.2).

w_i : Weight variable associated with the i^{th} reference point.

A weight variable 'w_i', with values constrained between zero and unity, is associated with each reference point 'i'. Therefore, the total number of weights is the same as the total number of reference points 'q'. A weight value at a reference point is considered to be representative of the belief value in the cause. However, the disadvantage of this formulation is that as the number of dimensions increase, the total number of weights in a network increase exponentially. This rapidly increases the number of unknown variables within the network and this is impractical, as it will not only slow down the system but also will require an excessively large training data set. For diagnostic problems the training data set is always limited. Therefore, the number of unknowns in the network need to be reduced to a minimum level.

6.3.1 Primary and Secondary Weight Values

After analysing the real casting data [5] and interaction with experts from the medical domain as well as the manufacturing industry, it was discovered that the belief value in the occurrence of a cause, is a linear combination of belief values in the cause when only one effect is assumed to have occurred at a time. For example, assume that cause 'c' is associated with two effects 'ξ^1' and 'ξ^2'. The belief values in the occurrence of effects 'ξ^1' and 'ξ^2' are scaled between –1 to +1 and the expected belief value in the occurrence of a cause 'c' will be between 0 and 1. Assuming a quadratic variation of belief values, corresponding to the occurrence of a cause along each axis representing belief values corresponding to the occurrence of effects, the total number of reference points 'q' will be equal to 9 ([using Equation 6.4, 6.7 = (2+1)*(2+1)]) as shown in Figure 6.3.

The observation, given in the previous paragraph, states that the belief value in the occurrence of a cause at reference point 5, can be represented as a linear function of the belief values in the occurrence of a cause at reference points 4 and 2 for the corresponding belief values in the effects 'ξ^1' and 'ξ^2'.

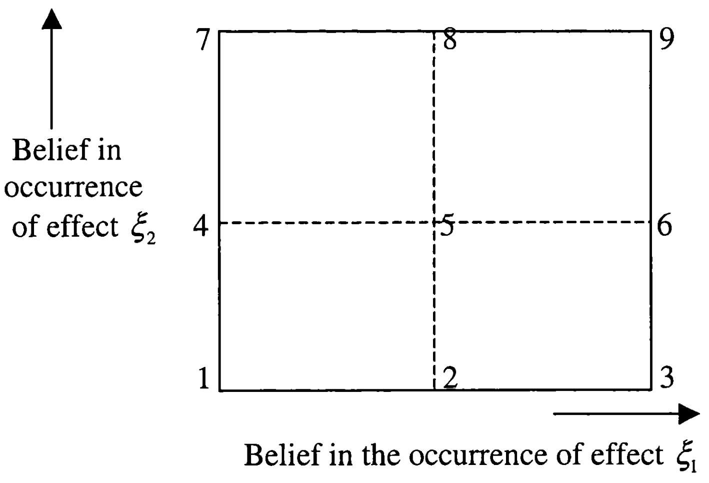

Belief in occurrence of effect ξ_2
1
2
3
4
5
6
7
8
9
Belief in the occurrence of effect ξ_1

Figure 6.3 Dependent and Independent Weight Values Associated with Reference Points 1 to 9.

This observation leads to an assumption that the weight values associated with reference points 1, 2, 3, 4 and 7 are independent and weight values associated with reference points 5, 6, 8 and 9 are dependent on the corresponding independent weight values. The independent weight values and their associated reference points are termed as the primary weight values and primary reference points respectively, whereas the dependent weight values are referred to as secondary weight values and are associated with secondary reference points. Thus, a secondary weight value can be represented as a linear combination of the corresponding primary weight values. It was also discovered that only one unknown is sufficient for this linear combination, which will be kept the same for all secondary weight variables.

With this implementation of encoding further known knowledge into the network, the number of unknowns requiring determination of optimal values reduces drastically. It will be seen later in this chapter, that the number of unknowns with this implementation are even less than the corresponding number of unknowns required with traditional neural network modelling.

The most important advantage of the research finding that the secondary weight values are a linear combination of the primary weight values is that the network can effectively learn on a small sized, noisy and conflicting data set and can attain exceptionally superior extrapolation abilities as compared with traditional neural networks. This advantage will be illustrated with the aid of an example later in this Chapter. The next subsection will complete the discussion of primary and secondary reference points.

6.3.2 Primary and Secondary Reference Points

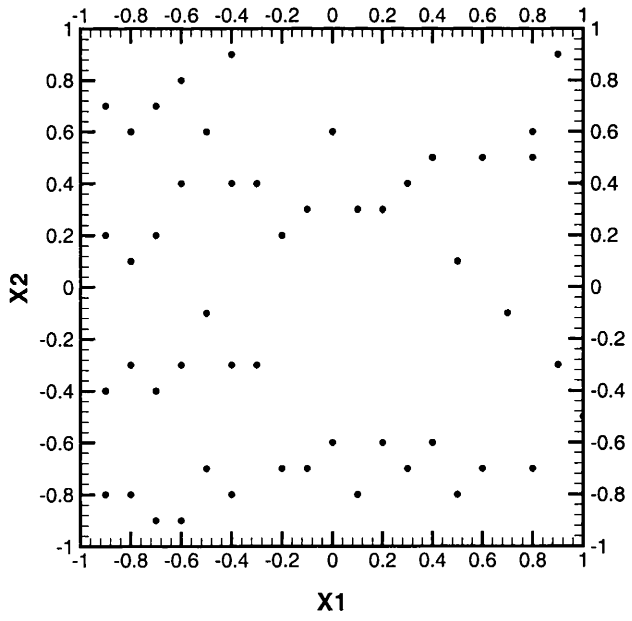
The reference points, which lie along the dimensional axes, are special points and are referred to as "primary reference points", i.e. the (n_j+1) reference points $\xi_0^j, \xi_1^j, \xi_2^j, \ldots, \xi_{n_j}^j$ that lie along the j^{th} dimension are the primary reference points. In terms of co-ordinates, a reference point is along the dimensional axis if one and only one of its co-ordinates (k_j) has a non-zero value and all other co-ordinates have a zero value. For a 'p' dimensional problem, the total number of primary weights is

$$\left[\left(\sum_{j=1}^{p} n_j + 1\right) - (p-1)\right] \tag{6.6}$$

During the learning, or training process, the optimal values for primary weights (constrained between zero and one) and coefficients used in the linear combination expression are determined based on the Gradient Descent algorithm as frequently used in traditional neural networks or any other optimisation techniques [7-8]. The secondary weights are also constrained between zero and one.

6.4 A Numerical Example of the Proposed Method

Consider an example where two effects, ξ^1 and ξ^2, are associated with a cause *c*. The numerical values between –1 and +1, associated with the variables ξ^1 and ξ^2, represent the belief value representing the occurrence/non-occurrence of the corresponding effects ξ^1 and ξ^2. Therefore, two-dimensional Lagrange Interpolation Polynomials $l_i\left(\xi^1,\xi^2\right)$ will be used for defining the hyper surface. A quadratic relationship is assumed between belief values for the effect ξ^1 and the cause *c*, and also for the effect ξ^2 and the cause *c*.



X1: Belief value in the occurrence of effect ξ^1.

X2: Belief value in the occurrence of effect ξ^2.

Figure 6.4 Data points used in the training data set of Table 6.1.

Training set:

Input X1	Input X2	Target Output
-0.9	-0.8	0.04
-0.9	-0.4	0.12
-0.9	0.2	0.23
-0.9	0.7	0.34
-0.8	-0.8	0.05
-0.8	-0.3	0.14
-0.8	0.1	0.21
-0.8	0.6	0.31
-0.7	-0.9	0.04
-0.7	-0.4	0.12
-0.7	0.2	0.23
-0.7	0.7	0.34
-0.6	-0.9	0.06
-0.6	-0.3	0.15
-0.6	0.4	0.28
-0.6	0.8	0.36
-0.5	-0.7	0.1
-0.5	-0.1	0.19
-0.5	0.6	0.33
-0.4	-0.8	0.11
-0.4	-0.3	0.17
-0.4	0.4	0.29
-0.4	0.9	0.41
-0.3	-0.3	0.19
-0.3	0.4	0.31
-0.2	-0.7	0.16
-0.2	0.2	0.29
-0.1	-0.7	0.19
-0.1	0.3	0.33
0	-0.6	0.23
0	0.6	0.41
0.1	-0.8	0.25
0.1	0.3	0.38
0.2	-0.6	0.3
0.2	0.3	0.41
0.3	-0.7	0.33
0.3	0.4	0.46
0.4	-0.6	0.38
0.4	0.5	0.51
0.5	-0.8	0.42
0.5	0.1	0.5
0.6	-0.7	0.47
0.6	0.5	0.59
0.7	-0.1	0.57
0.8	-0.7	0.59
0.8	0.5	0.69
0.8	0.6	0.70
0.9	-0.3	0.67
0.9	0.9	0.79
1	-0.5	0.72
1	0.4	0.78

$\longleftarrow$ Table 6.1

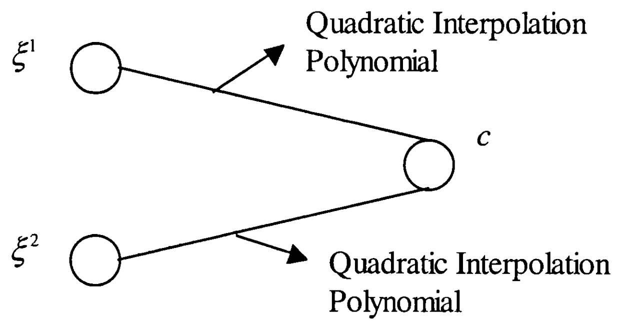

ξ^1
Quadratic Interpolation Polynomial
c
ξ^2
Quadratic Interpolation Polynomial

Figure 6.5: Schematic diagram of two effects - one cause relationship.

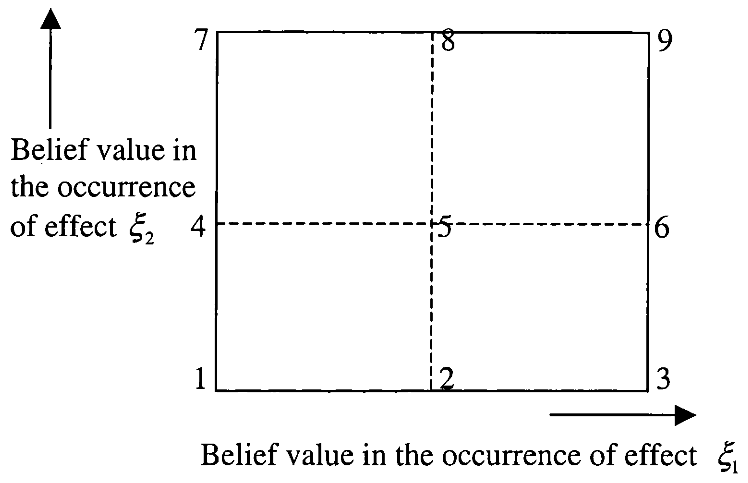

7
8
9
Belief value in the occurrence of effect ξ_2
4
5
6
1
2
3
Belief value in the occurrence of effect ξ_1

Figure 6.6: Two dimensional input space describing ξ^1 and ξ^2 axes. Numbers 1 to 9 denote equidistant reference points. As a result of the quadratic relationship, (n_1 and n_2 equal to 2), three equidistant reference points are used along each dimension. Using Equation (6.4), it can be seen that the total number of reference points is 9.

Using Equation (6.6), it can be seen that the total number of primary reference points is 5. These points are also indicated in the following table (Table 6.2), which shows the co-ordinates of all nine-reference points in various forms.

Weights associated with primary reference points 1, 2, 3, 4 and 7 are primary weights and are also independent parameters. The secondary weight values at locations 5, 6, 8 and 9 are expressed as a linear combination of the primary weights and in particular

$$w_5 = \frac{C(w_2 + w_4)}{2} \tag{6.7}$$

$$w_6 = \frac{C(w_3 + w_4)}{2} \tag{6.8}$$

$$w_8 = \frac{C(w_2 + w_7)}{2} \tag{6.9}$$

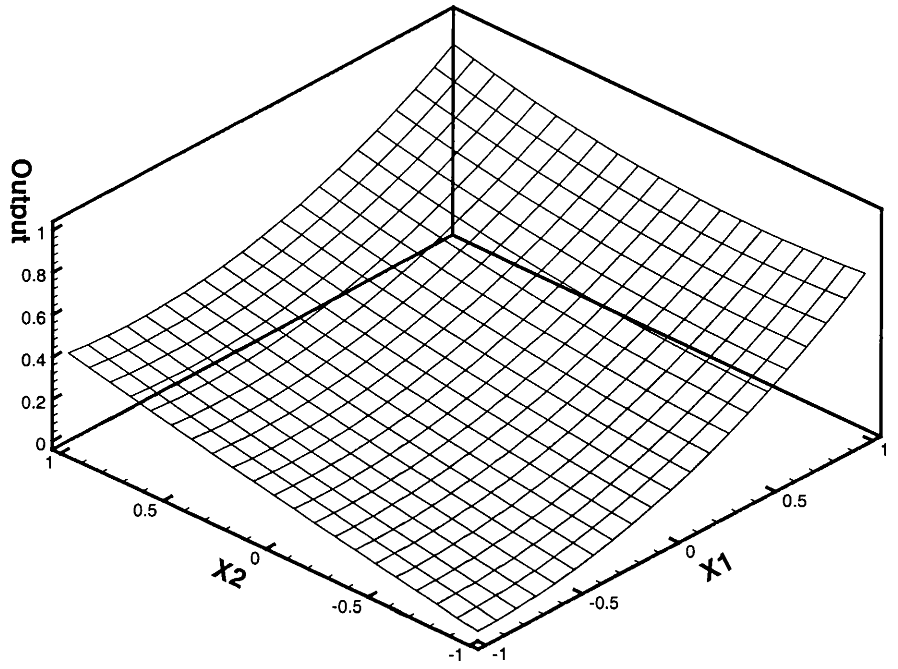
$$\text{and } w_9 = \frac{C(w_3 + w_7)}{2} \tag{6.10}$$

Thus, the independent parameters, which are passed on to the optimisation algorithm during the learning process are w_1, w_2, w_3, w_4, w_7 and the constant *C*.

Reference Points $(i = 1\,to\,q)$	Coordinates in terms of k_j	Coordinates in terms of ξ^j	Coordinates in terms of actual numerical values for ξ^j's
1.	(0,0) Primary	(ξ_0^1, ξ_0^2)	(-1, -1)
2.	(1,0) Primary	(ξ_1^1, ξ_0^2)	(0, -1)
3.	(2,0) Primary	(ξ_2^1, ξ_0^2)	(+1, -1)
4.	(0,1) Primary	(ξ_0^1, ξ_1^2)	(-1, 0)
5.	(1,1) Secondary	(ξ_1^1, ξ_1^2)	(0, 0)
6.	(2,1) Secondary	(ξ_2^1, ξ_1^2)	(1, 0)
7.	(0,2) Primary	(ξ_0^1, ξ_2^2)	(-1, +1)
8.	(1,2) Secondary	(ξ_1^1, ξ_2^2)	(0, +1)
9.	(2,2) Secondary	(ξ_2^1, ξ_2^2)	(+1, +1)

Table 6.2: Coordinates of Primary and Secondary Reference Points.

The proposed network is trained with the training data given in Figure 6.4 and tabulated in Table 6.1, and the output surface, representing the belief values in the occurrence of cause for various belief values in the occurrence of effects ξ^1 and ξ^2, is shown in Figure 6.7.



X1: Belief value in the occurrence of effect ξ^1.

X2: Belief value in the occurrence of effect ξ^2.

Output: Belief value in the occurrence of the associated cause c.

Figure 6.7: The optimal surface representing belief values in the occurrence of cause c.

For this example, the belief value in the occurrence of cause c is calculated for a belief value for the first effect ξ^1 of 0.5 and a belief value for the second effect ξ^2 of –0.5, using the final (i.e. optimal) weights. The procedure is as follows.

The belief value, representing the extent of the occurrence of a cause, for a given belief value ξ^1 equal to 0.5 (representing the strength of effect ξ^1) and a given belief value ξ^2 equal to –0.5 (representing the strength of effect ξ^2) is calculated as:

The belief value in the occurrence of cause = $\sum_{i=1}^{9} w_i l_i (0.5,-0.5)$

Using Equations (6.2) and (6.3), Lagrange Interpolation Polynomials are constructed and then evaluated at (0.5, -0.5) at all reference points.

Lagrange Interpolation Polynomial for reference point 1:

$$l_1\left(\xi^1,\xi^2\right)=l_0^2\left(\xi^1\right)*l_0^2\left(\xi^2\right)$$
$$=\left(\frac{\xi^1-\xi_1^1}{\xi_0^1-\xi_1^1}*\frac{\xi^1-\xi_2^1}{\xi_0^1-\xi_2^1}\right)*\left(\frac{\xi^2-\xi_1^2}{\xi_0^2-\xi_1^2}*\frac{\xi^2-\xi_2^2}{\xi_0^2-\xi_2^2}\right)$$
$$=\frac{1}{4}\left(\xi^1\right)\left(\xi^1-1\right)\left(\xi^2\right)\left(\xi^2-1\right)$$

$l_1(0.5,-0.5)=-0.0469$

Lagrange Interpolation Polynomial for reference point 2:

$$l_2\left(\xi^1,\xi^2\right)=l_1^2\left(\xi^1\right)*l_0^2\left(\xi^2\right)$$
$$=\left(\frac{\xi^1-\xi_0^1}{\xi_1^1-\xi_0^1}*\frac{\xi^1-\xi_2^1}{\xi_1^1-\xi_2^1}\right)*\left(\frac{\xi^2-\xi_1^2}{\xi_0^2-\xi_1^2}*\frac{\xi^2-\xi_2^2}{\xi_0^2-\xi_2^2}\right)$$
$$=-\frac{1}{2}\left(\xi^1+1\right)\left(\xi^1-1\right)\left(\xi^2\right)\left(\xi^2-1\right)$$

$l_2(0.5,-0.5)=0.2813$

Lagrange Interpolation Polynomial for reference point 3:

$$l_3\left(\xi^1,\xi^2\right)=l_2^2\left(\xi^1\right)*l_0^2\left(\xi^2\right)$$
$$=\left(\frac{\xi^1-\xi_0^1}{\xi_2^1-\xi_0^1}*\frac{\xi^1-\xi_1^1}{\xi_2^1-\xi_1^1}\right)*\left(\frac{\xi^2-\xi_1^2}{\xi_0^1-\xi_1^2}*\frac{\xi^2-\xi_2^2}{\xi_0^2-\xi_2^1}\right)$$
$$=\frac{1}{4}\left(\xi^1+1\right)\left(\xi^1\right)\left(\xi^2\right)\left(\xi^2-1\right)$$

$l_3(0.5,-0.5)=0.1406$

Lagrange Interpolation Polynomial for reference point 4:

$$l_4\left(\xi^1,\xi^2\right)=l_0^2\left(\xi^1\right)*l_1^2\left(\xi^2\right)$$
$$=\left(\frac{\xi^1-\xi_1^1}{\xi_0^1-\xi_1^1}*\frac{\xi^1-\xi_2^1}{\xi_0^1-\xi_2^1}\right)*\left(\frac{\xi^2-\xi_0^2}{\xi_1^2-\xi_0^2}*\frac{\xi^2-\xi_2^2}{\xi_1^2-\xi_2^2}\right)$$
$$=-\frac{1}{2}\left(\xi^1\right)\left(\xi^1-1\right)\left(\xi^2+1\right)\left(\xi^2-1\right)$$

$l_4(0.5,-0.5)=-0.0938$

Lagrange Interpolation Polynomial for reference point 5:

$$l_5(\xi^1,\xi^2) = l_1^2(\xi^1) * l_1^2(\xi^2)$$

$$= \left(\frac{\xi^1 - \xi_0^1}{\xi_1^1 - \xi_0^1} * \frac{\xi^1 - \xi_2^1}{\xi_1^1 - \xi_2^1} \right) * \left(\frac{\xi^2 - \xi_0^2}{\xi_1^2 - \xi_0^2} * \frac{\xi^2 - \xi_2^2}{\xi_1^2 - \xi_2^2} \right)$$

$$= (\xi^1 + 1)(\xi^1 - 1)(\xi^2 + 1)(\xi^2 - 1)$$

$$l_5(0.5,-0.5) = 0.5625$$

Lagrange Interpolation Polynomial for reference point 6:

$$l_6(\xi^1,\xi^2) = l_2^2(\xi^1) * l_1^2(\xi^2)$$

$$= \left(\frac{\xi^1 - \xi_0^1}{\xi_2^1 - \xi_0^1} * \frac{\xi^1 - \xi_1^1}{\xi_2^1 - \xi_1^1} \right) * \left(\frac{\xi^2 - \xi_0^2}{\xi_1^2 - \xi_0^2} * \frac{\xi^2 - \xi_2^2}{\xi_1^2 - \xi_2^2} \right)$$

$$= -\frac{1}{2}(\xi^1 + 1)(\xi^1)(\xi^2 + 1)(\xi^2 - 1)$$

$$l_6(0.5,-0.5) = 0.2813$$

Lagrange Interpolation Polynomial for reference point 7:

$$l_7(\xi^1,\xi^2) = l_0^2(\xi^1) * l_2^2(\xi^2)$$

$$= \left(\frac{\xi^1 - \xi_1^1}{\xi_0^1 - \xi_1^1} * \frac{\xi^1 - \xi_2^1}{\xi_0^1 - \xi_2^1} \right) * \left(\frac{\xi^2 - \xi_0^2}{\xi_2^2 - \xi_0^2} * \frac{\xi^2 - \xi_1^2}{\xi_2^2 - \xi_1^2} \right)$$

$$= \frac{1}{4}(\xi^1)(\xi^1 - 1)(\xi^2 + 1)(\xi^2)$$

$$l_7(0.5,-0.5) = 0.0156$$

Lagrange Interpolation Polynomial for reference point 8:

$$l_8(\xi^1,\xi^2) = l_1^2(\xi^1) * l_2^2(\xi^2)$$

$$= \left(\frac{\xi^1 - \xi_0^1}{\xi_1^1 - \xi_0^1} * \frac{\xi^1 - \xi_2^1}{\xi_1^1 - \xi_2^1} \right) * \left(\frac{\xi^2 - \xi_0^2}{\xi_2^2 - \xi_0^2} * \frac{\xi^2 - \xi_1^2}{\xi_2^2 - \xi_1^2} \right)$$

$$= -\frac{1}{2}(\xi^1 + 1)(\xi^1 - 1)(\xi^2 + 1)(\xi^2)$$

$$l_8(0.5,-0.5) = -0.0938$$

Lagrange Interpolation Polynomial for reference point 9:

$$l_9(\xi^1,\xi^2)=l_2^2(\xi^1)*l_1^2(\xi^2)$$

$$=\left(\frac{\xi^1-\xi_0^1}{\xi_2^1-\xi_0^1}*\frac{\xi^1-\xi_1^1}{\xi_2^1-\xi_1^1}\right)*\left(\frac{\xi^2-\xi_0^2}{\xi_2^2-\xi_0^2}*\frac{\xi^2-\xi_1^2}{\xi_2^2-\xi_1^2}\right)$$

$$=\frac{1}{4}(\xi^1+1)(\xi^1)(\xi^2+1)(\xi^2)$$

$$l_9(0.5,-0.5)=-0.0469$$

Assuming that the independent parameters, which are passed on to the optimisation algorithm during the learning process have final values as $w_1=0.0084$, $w_2=0.1972$, $w_3=0.4179$, $w_4=0.1924$, $w_7=0.7359$ and $C=1.5656$, then using Equations (6.7), (6.8), (6.9) and (6.10) $w_5=0.3050$, $w_6=0.4778$, $w_8=0.7304$ and $w_9=0.9032$.

Belief value in the cause = $\sum_{i=1}^{9} w_i l_i(0.5,-0.5)$

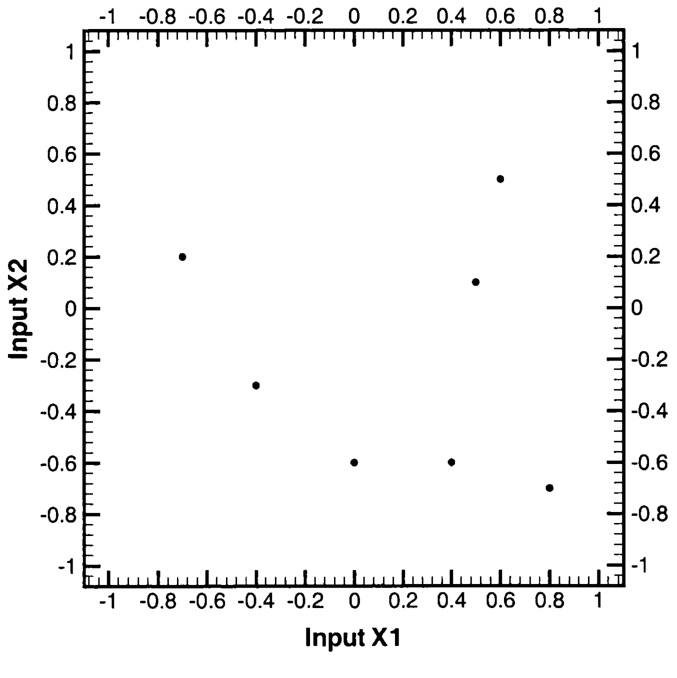
$=0.3024$

6.5 Comparison of the Learning Abilities of the Lagrange Interpolation Polynomial Regression network with a Multi Layer Neural Network

Code for the Lagrange Interpolation Polynomial Regression Network, has been developed by the author using C++ programming Language. During the training process, the optimal values for primary weights and coefficients used in the linear combination expression are determined based on the Gradient Descent algorithm as frequently used in traditional neural networks. Three representative cases are designed to test the performance of the Lagrange Interpolation Polynomial Regression network and a multi layer neural network. The first test case assesses the learning ability of both networks on a small and noisy training data. The second and third test cases compare the learning performance on an unevenly spaced training data. Comments are made on the extrapolation abilities of both networks. In all test cases, the training data set is generated from the output belief variation as shown in Figure 6.7, corresponding to belief values giving the extent of occurrence of the effects ξ^1 and ξ^2. In creating the training data set, a few data points are chosen randomly and then a maximum of 20% noise is also randomly added to the training data set values to generate the corresponding target output.

Case 1: Extremely limited but approximately evenly distributed noisy training data.

A very limited training data set is chosen for this test case, which is shown in Table 6.3 and graphically plotted in Figure 6.8. For a traditional neural network then a best least squares fit surface to the training data requires that the training data set has a greater number of data points than the number of unknowns in the network. In a multi layer neural network, the number of unknowns increases as the number of hidden nodes increase. For example, for a two input – one output neural network, with three or five hidden nodes and one output node, will require at least 13 or 21 data points, respectively, in the training data set. This example has only seven data points. Therefore, it is a limitation of the multi-layer neural network that it cannot work with a very small training data set. However, the Lagrange Interpolation Polynomial Regression network has only six unknowns and hence this network can train even with very small data sets.



X1: Belief value in the occurrence of effect ξ^1.

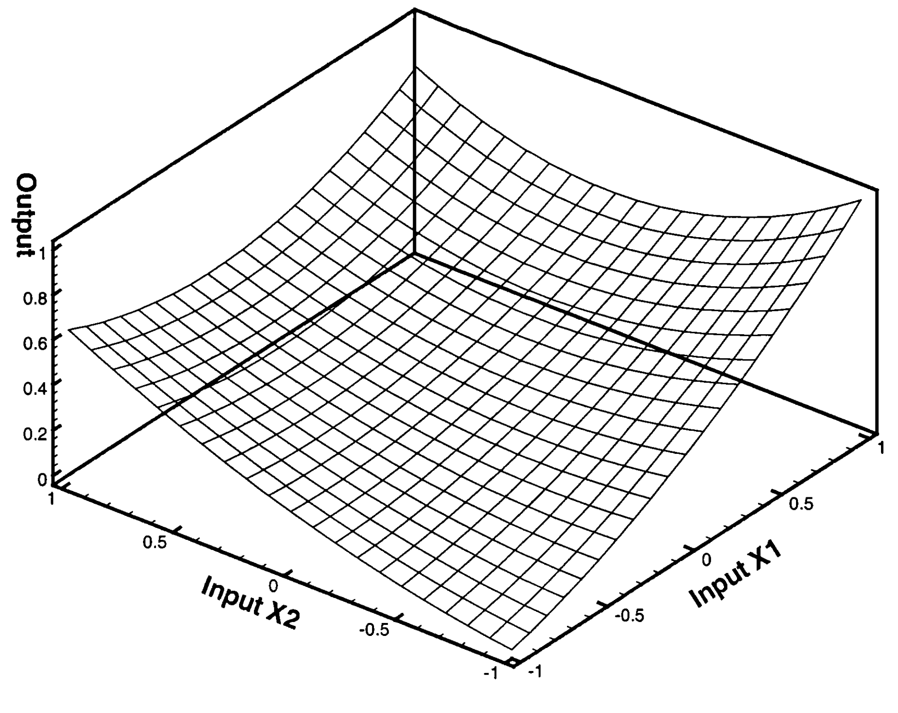
X2: Belief value in the occurrence of effect ξ^2.

Figure 6.8: Data points used in the training data set as tabulated in Table 6.3.

Training set:

Input X1	Input X2	Target Output
-0.7	0.2	0.4
-0.4	-0.3	0.19
0	-0.6	0.06
0.4	-0.6	0.56
0.5	0.1	0.41
0.6	0.5	0.51
0.8	-0.7	0.36

Table 6.3 The training data set used for test Case 1.



X1: Belief value in the occurrence of effect ξ^1.

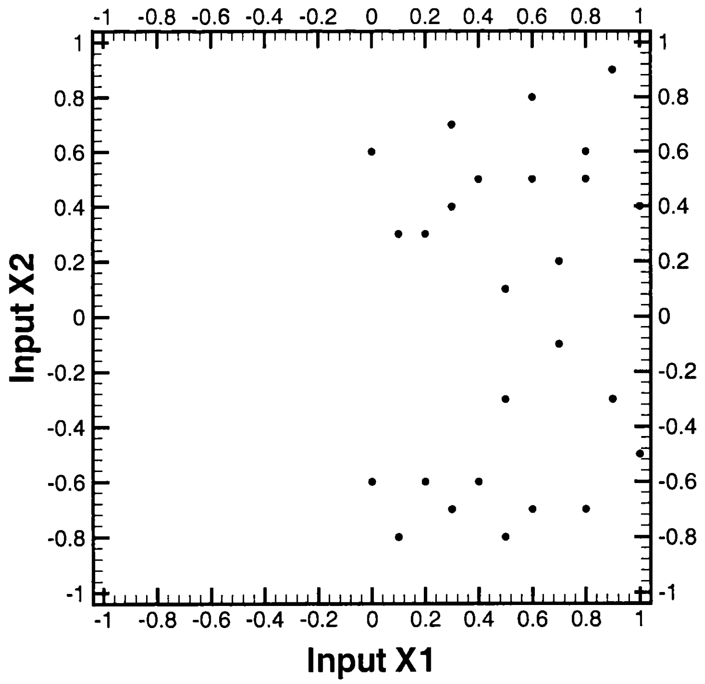
X2: Belief value in the occurrence of effect ξ^2.

Output: Belief value in the occurrence of the associated cause *c*.

Figure 6.9: Output Surface from the Lagrange Interpolation Polynomial Regression network Trained with the Training Data of Table 6.3.

Case 2: Noisy training data set with values constrained to high values for ξ^1 within the input space.

The training data set is shown in Figure 6.10 and tabulated in Table 6.4. A neural network with two input nodes, five hidden nodes and one output node is constructed and trained on the training data set with a learning rate equal to 0.4 and with a target error of 0.01. The solution converged in 10139 epochs using the gradient descent algorithm. The optimal surface, representing the output belief values in the occurrence of the cause '*c*' obtained by the neural network, is shown in Figure 6.12, whereas the optimal surface obtained by the Lagrange Interpolation Polynomial Regression network is shown in Figure 6.11. Comparing these surfaces with the original surface (Figure 6.7), it can be observed that the Lagrange Interpolation Polynomial Regression network outperforms the neural network in its extrapolation abilities. This has become possible because of the assumed interrelationship between the secondary and primary weight values.



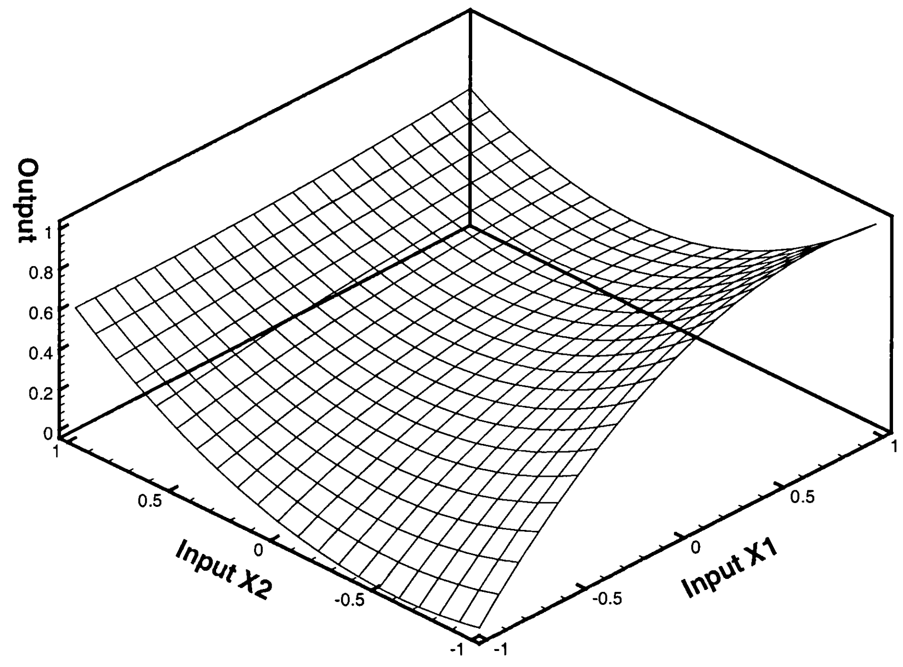
X1: Belief value in the occurrence of effect ξ^1.

X2: Belief value in the occurrence of effect ξ^2.

Figure 6.10: Data points used in the training data set as tabulated in Table 6.4.

Training set:		
Input X1	Input X2	Target Output
0	-0.6	0.40
0	0.6	0.43
0.1	-0.8	0.08
0.1	0.3	0.55
0.2	-0.6	0.21
0.2	0.3	0.32
0.3	-0.7	0.11
0.3	0.4	0.54
0.3	0.7	0.32
0.4	-0.6	0.42
0.4	0.5	0.12
0.5	-0.8	0.27
0.5	-0.3	0.34
0.5	0.1	0.33
0.6	-0.7	0.70
0.6	0.5	0.47
0.6	0.8	0.59
0.7	-0.1	0.51
0.7	0.2	0.27
0.8	-0.7	0.21
0.8	0.5	0.80
0.8	0.6	0.53
0.9	-0.3	0.24
0.9	0.9	0.60
1	-0.5	0.76
1	0.4	0.88

Table 6.4: The training data set used for test Case 2.

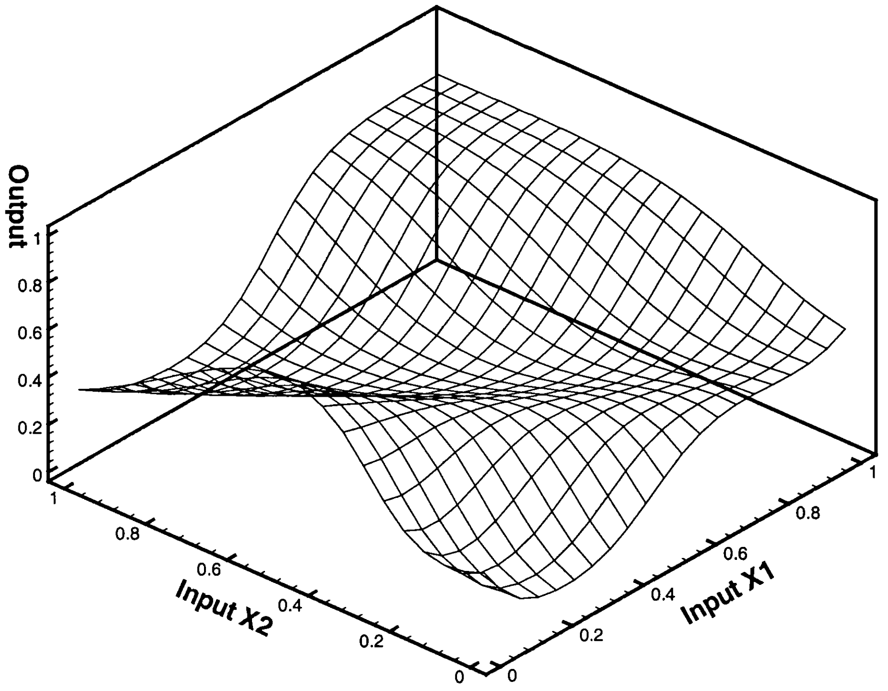


X1: Belief value in the occurrence of effect ξ^1.

X2: Belief value in the occurrence of effect ξ^2.

Output: Belief value in the occurrence of the associated cause *c*.

Figure 6.11: Output Surface from the Lagrange Interpolation Polynomial Regression network Trained using the Training Data of Table 6.4.



X1: Belief value in the occurrence of effect ξ^1.

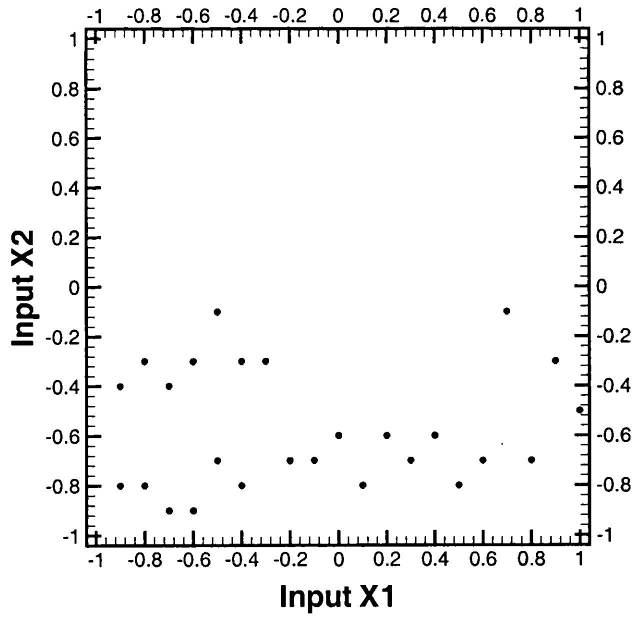
X2: Belief value in the occurrence of effect ξ^2.

Output: Belief value in the occurrence of the associated cause *c*.

Figure 6.12: Output Surface from the Multi Layer Neural Network Trained using the Training Data of Table 6.4.

Case 3: Noisy training data set with values constrained to low values for ξ^2 within the input space.

The training data set is shown in Figure 6.13 and tabulated in Table 6.5. A neural network with two input nodes, five hidden nodes and one output node is constructed and trained on the training data set with a learning rate equal to 0.4 and with a target error of 0.01. The solution converged in 15893 epochs using the gradient descent algorithm. The optimal surface, representing the output belief values in the occurrence of the cause '*c*' obtained by the neural network, is shown in Figure 6.15, whereas the optimal surface obtained by the Lagrange Interpolation Polynomial Regression network is shown in Figure 6.14. Comparing these surfaces with the original surface (Figure 6.7), it can be observed that the Lagrange Interpolation Polynomial Regression network outperforms the neural network in its extrapolation abilities due to the assumed interrelationship between the secondary and primary weight values. This strengthens the observations made in the previous test case.



X1: Belief value in the occurrence of effect ξ^1.

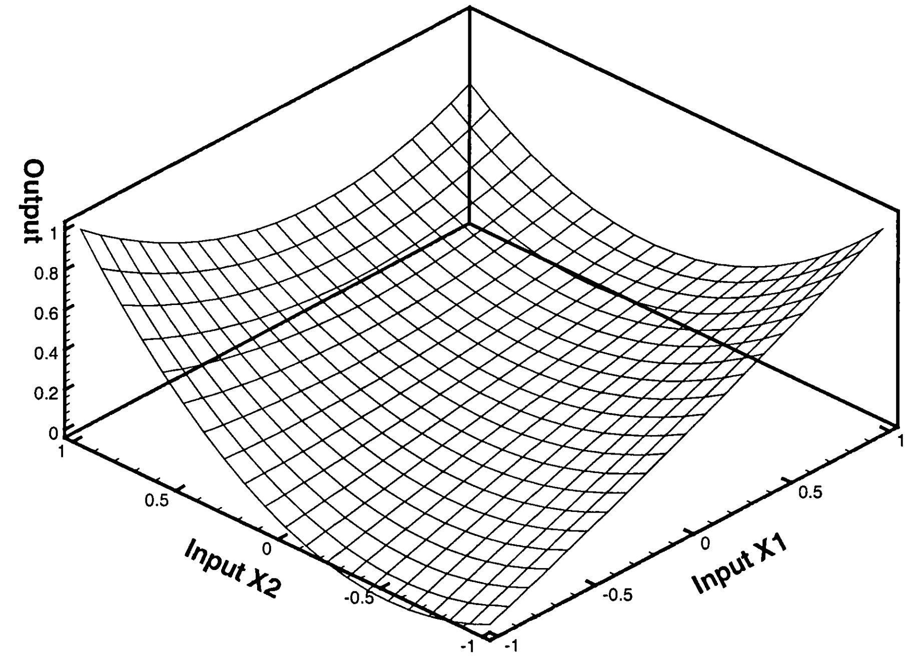
X2: Belief value in the occurrence of effect ξ^2.

Figure 6.13: Data points used in the training data set as tabulated in Table 6.5.

Training set:

Input X1	Input X2	Target Output
-0.9	-0.8	0.22
-0.9	-0.4	0.14
-0.8	-0.8	-0.12
-0.8	-0.3	0.31
-0.7	-0.9	-0.05
-0.7	-0.4	0.04
-0.6	-0.9	-0.16
-0.6	-0.3	0.23
-0.5	-0.7	-0.09
-0.5	-0.1	0.22
-0.4	-0.8	-0.29
-0.4	-0.3	0.02
-0.3	-0.3	0.08
-0.2	-0.7	-0.01
-0.1	-0.7	0.41
0	-0.6	0.11
0.1	-0.8	0.20
0.2	-0.6	0.24
0.3	-0.7	-0.01
0.4	-0.6	0.01
0.5	-0.8	0.53
0.6	-0.7	0.30
0.7	-0.1	0.15
0.8	-0.7	0.39
0.9	-0.3	0.71
1	-0.5	0.81

Table 6.5: The training data set used for test Case 3.

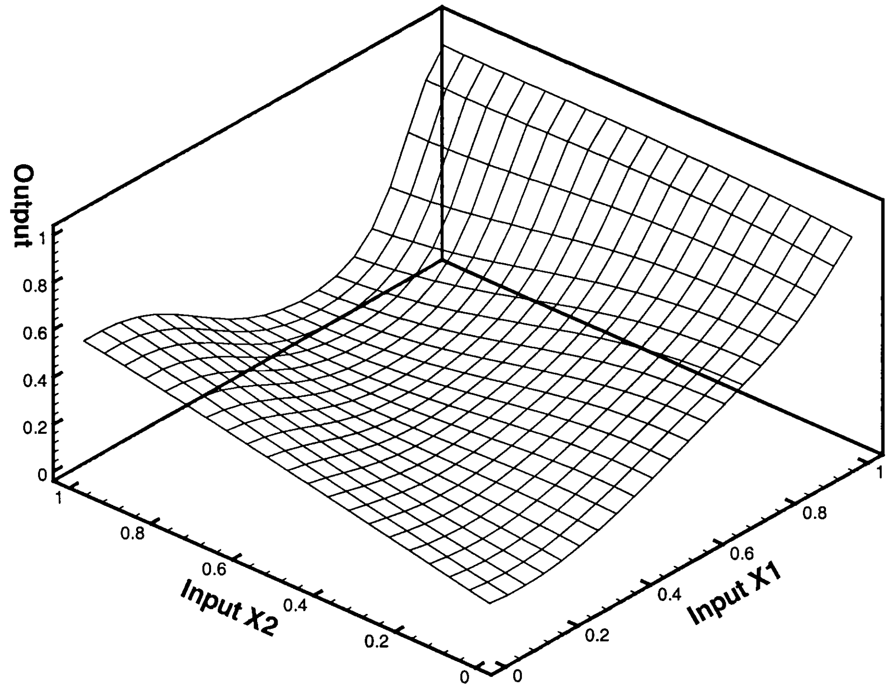


X1: Belief value in the occurrence of effect ξ^1.

X2: Belief value in the occurrence of effect ξ^2.

Output: Belief value in the occurrence of the associated cause c.

Figure 6.14: Output Surface from the Lagrange Interpolation Polynomial Regression network Trained using the Training Data of Table 6.5.



X1: Belief value in the occurrence of effect ξ^1.

X2: Belief value in the occurrence of effect ξ^2.

Output: Belief value in the occurrence of the associated cause c.

Figure 6.15: Output Surface from the Multi Layer Neural Network Trained using the Training Data of Table 6.5.

After observing the outputs from both the networks, for both examples, we see that the Lagrange Interpolation Polynomial Regression networks output is similar to that of Figures 6.7 (trained with all the data) and 6.9 (trained with a very small amount of original data), but the output from the traditional neural network shows that the output depends on the location and number of data points as well as the target output of each data point in the training data set.

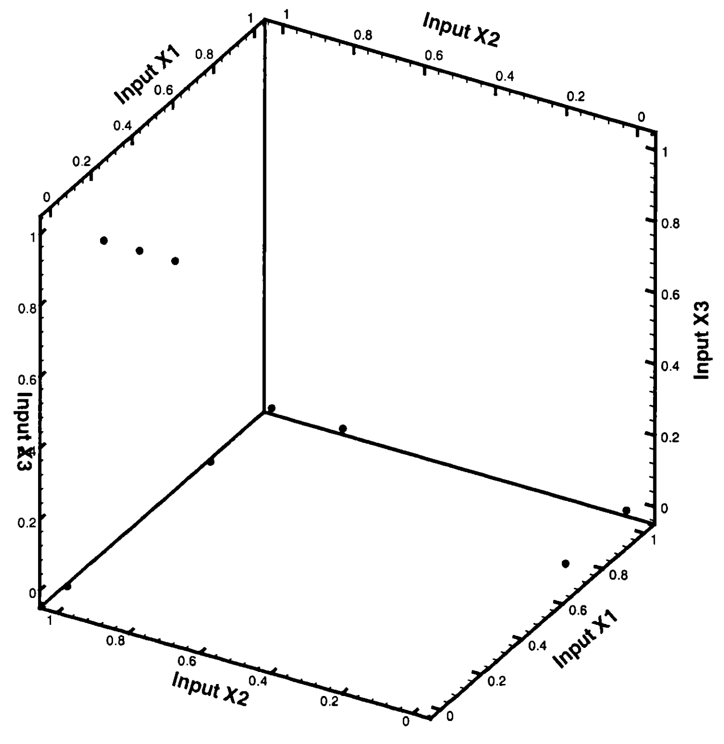
6.6 Comparison of the Lagrange Interpolation Polynomial Regression Network with a Multi Layer Neural Network on a Real Data Set

The author has compared the outputs from both the Lagrange Interpolation Polynomial Regression network and a multi layer neural network on a real data set. The data was collected from 'Kaye Preistigne' – a pressure die casting foundry. A total of 14 defects were identified and associated with 43 process, material or design parameters. The data was collected for similar components over a period of one year. A total of 60 representative examples were finalised. A belief value in the occurrence of defects was calculated as described previously in Section 2.4 (Chapter 2). The corresponding belief values representing the occurrence and non-occurrence of associated process, design and material parameters were given by the experts in the foundry. To show the graphical variation of belief surfaces learnt by neural network and the Lagrange Interpolation Polynomial Regression network method, three defects identified as 'Porosity', 'Mismakes' and 'Dimensional' were chosen and are represented as defect A, defect B and defect C in Table 6.6. Sixteen associated process, material and design parameters were identified to create a neural network with three input nodes corresponding to defects 'defect A', 'defect B' and 'defect C' and sixteen output nodes corresponding to the sixteen process, material and design parameters. The belief values which were used in a training data set are shown in Table 6.6. The neural network was chosen with five hidden nodes, and a quadratic variation between input and output relationship was assumed in the Lagrange Interpolation Polynomial Regression network. Both networks were trained on the training data set as shown in Figure 6.16. Codes for both, the Lagrange Interpolation Polynomial Regression Network and a multi layer neural network, have been rewritten by the author in Matlab. The performance of both the networks was tested and compared using faster optimisation algorithms such as the Quasi-Newton. Both networks achieved the target error of 0.01 and seemed to have learnt the training data set. However, a remarkable difference can be observed in the shape of the belief surface. The belief surface has been plotted for the three causes:

a)	Number 8:	The position of gate.
b)	Number 12:	Kind and area of die venting.
c)	Number 16:	The position of overflow.

"The position of gate" (cause number 8) influences the occurrence of "Porosity" (defect A) and "Mismakes" (defect B) whereas "Kind and area of die venting" (cause number 12) and "The position of overflow" (cause number 16) influence the occurrence of all three defects, "Porosity" (defect A), "Mismakes" (defect B) and "Dimensional" (defect C). Figure 6.17 shows the variation in belief values in the occurrence of "The position of gate" given belief values for the three input values (-0.8, 0 and +0.8) for defect C i.e. "Dimensional" and for all belief values for defects "Porosity" and "Mismakes" using the Lagrange Interpolation Polynomial Regression network, and the corresponding neural network results are shown in Figure 6.18. Similarly, results are shown for other two causes: "Kind and area of die venting" and "The position of overflow" in Figures 6.19 to 6.22.

It can be easily observed that although the target error value of 0.01 was the same for both techniques, the belief variation surface learned by the neural network is incorrect in almost all cases, particularly in regions where insufficient training data was available.



X1: Belief value in the occurrence of defect "Porosity".

X2: Belief value in the occurrence of defect "Mismakes".

X3: Belief value in the occurrence of defect "Dimensional".

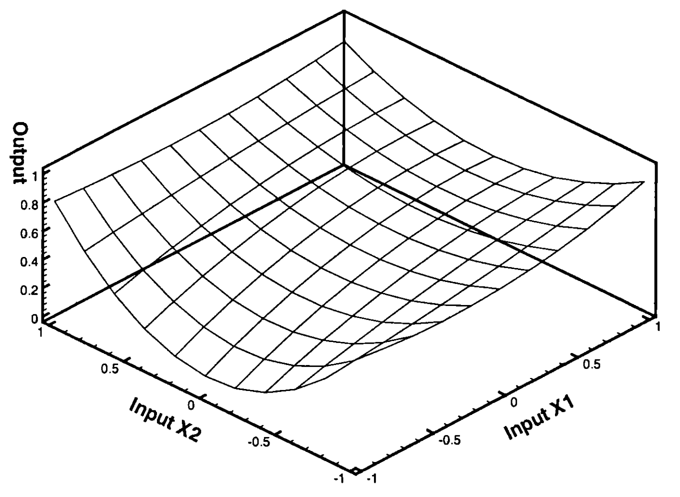
Figure 6.16: Data points used in the training data set as tabulated in Table 6.6.

Defects:	Defect A	Defect B	Defect C
Strength of Defects:			
	1	0	0
	1	0	0
	0.7	0	0
	1	1	0
	0.7	1	0
	1	0.8	0
	0.7	1	0
	0	1	0
	0	1	0
	0	1	0
	0	0.8	1
	0	0.7	1
	0	0.9	1

Output node Numbers:	1	2	3	4	5	6	7	8
Target Output Values:								
	0.8	0	0	0	1	1	1	1
	0.8	0	0	0	1	1	1	1
	0	0	0	0	1	1	1	0.7
	0.8	0.7	0	0	1	1	1	1
	0.8	0.7	0	0	1	1	1	1
	0.8	0.7	0	0	1	1	1	1
	0.8	0.7	0	0	1	1	1	1
	0	0	0	0	0	0	0	0.8
	0	0	0	0	0	0	0	0.8
	0	0	0	0	0	0	0	0.9
	0	0.7	1	0.7	0	0	0	0.7
	0	0.7	1	0.7	0	0	0	0
	0	0.8	1	0.7	0	0	0	0.8

Output Node Numbers:	9	10	11	12	13	14	15	16
Target Output Values:								
	1	0.9	1	0.8	0	0	0	0.9
	1	0.9	1	0.8	0	0	0	0.9
	0.8	0.7	0.8	0	0	0	0	0
	1	1	1	0.8	0.8	0.7	0	0.9
	1	1	1	0.8	0.8	0.7	0	0.8
	1	1	1	0.8	0.8	0.7	0	0.9
	1	1	1	0.8	0.8	0.7	0	0.8
	0	0.9	0.7	0	0.7	0	0	0
	0	0.9	0.7	0	0.7	0	0	0
	0	0.9	0.7	0	0.8	0	0	0
	0	0.9	0	0.7	0.7	0	0	0
	0	0.9	0	0	0	0	0	0
	0	1	0.7	0.7	0.7	0	0	0

Table 6.6. The training data set with target output values for the input defects plotted in Figure 6.16.



X1: Belief value in the occurrence of "Porosity" defect.

X2: Belief value in the occurrence of defect "Mismake" defect.

Output: Belief value in the occurrence of "The position of gate"

Figure 6.17a:
For an input value of –0.8 for the "Dimensional" defect.

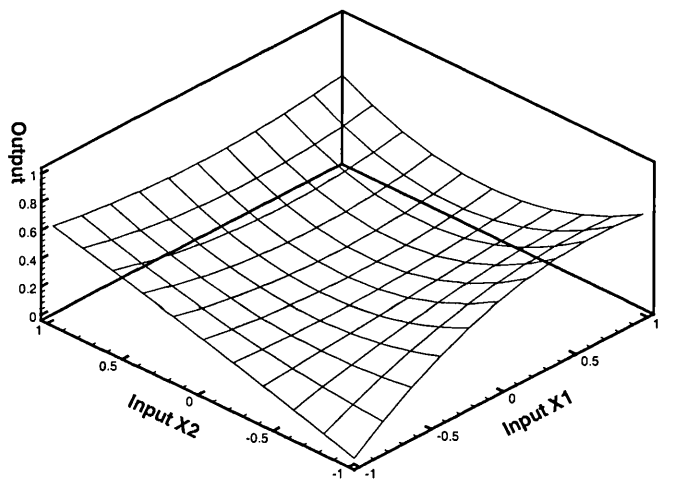


Figure 6.17b:
For an input value of 0 for the "Dimensional" defect.

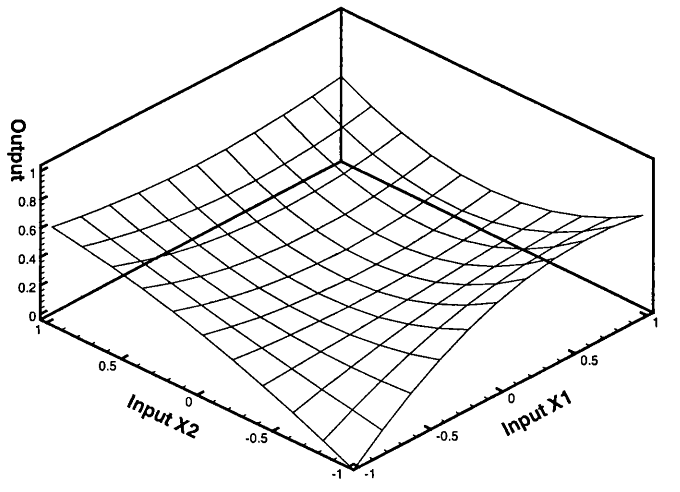
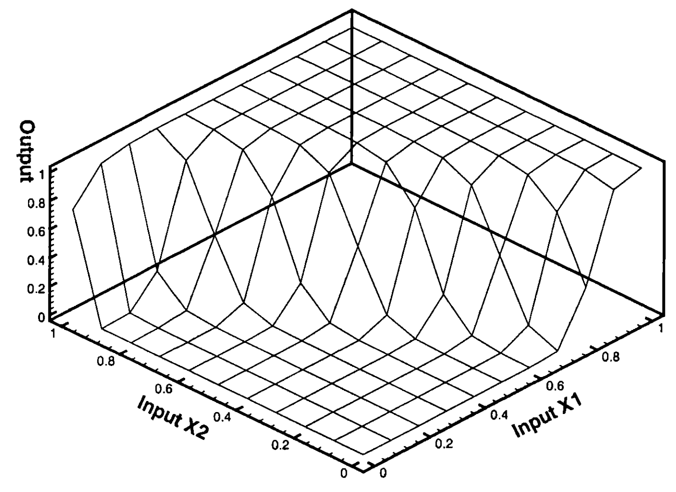


Figure 6.17c:
For an input value of +0.8 for the "Dimensional" defect.

Figure 6.17: Output Surfaces generated by the Lagrange Interpolation Polynomial Regression Network for a cause "The Position of gate".



X1: Belief value in the occurrence of "Porosity" defect.

X2: Belief value in the occurrence of defect "Mismake" defect.

Output: Belief value in the occurrence of "The position of gate"

Figure 6.18a:
For an input value of 0.1 for the "Dimensional" defect.

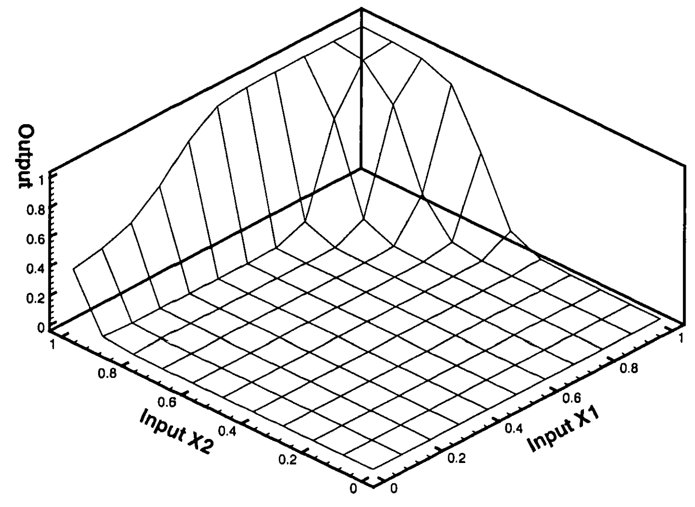


Figure 6.18b:
For an input value of 0.5 for the "Dimensional" defect.

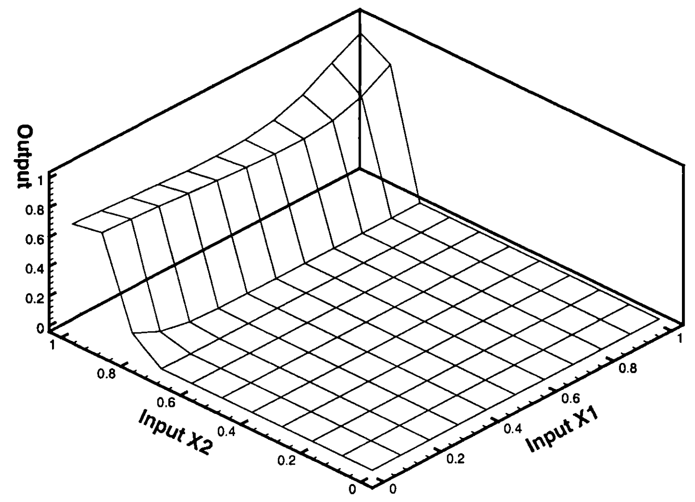
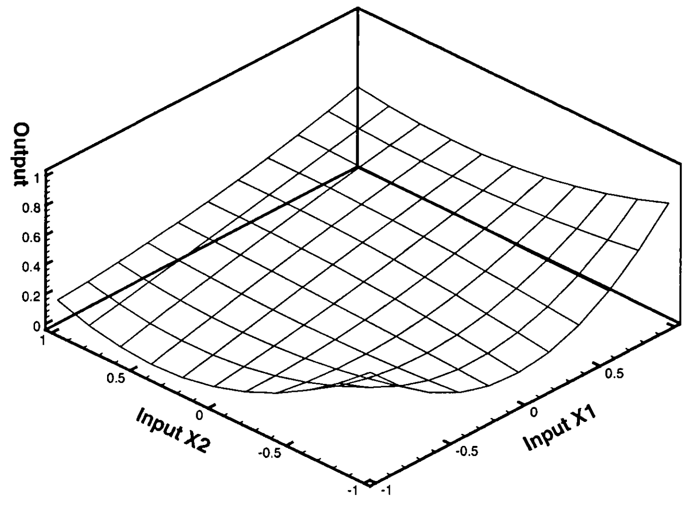


Figure 6.18c:
For an input value of 0.9 for the "Dimensional" defect.

Figure 6.18: Output Surfaces generated by the traditional neural network for a cause "The position of gate".



X1: Belief value in the occurrence of "Porosity" defect.

X2: Belief value in the occurrence of defect "Mismake" defect.

Output: Belief value in the occurrence of "Kind and area of die venting".

Figure 6.19a:
For an input value of –0.8 for the "Dimensional" defect.

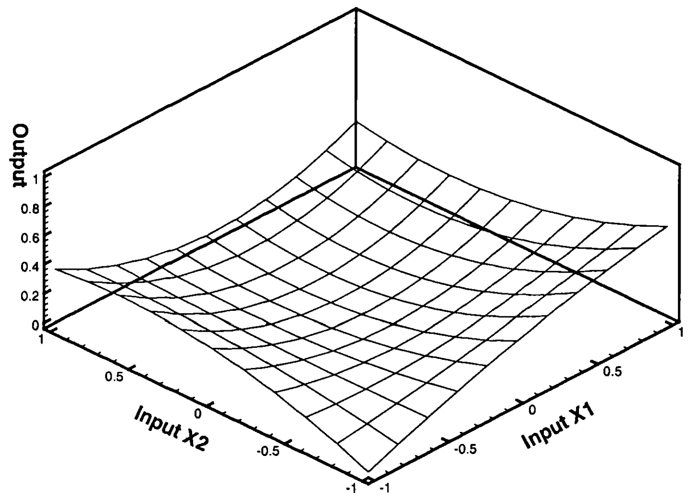


Figure 6.19b:
For an input value of 0 for the "Dimensional" defect.

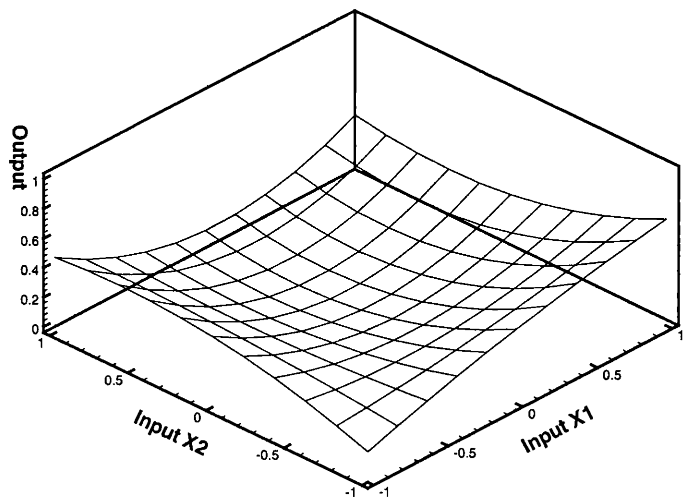
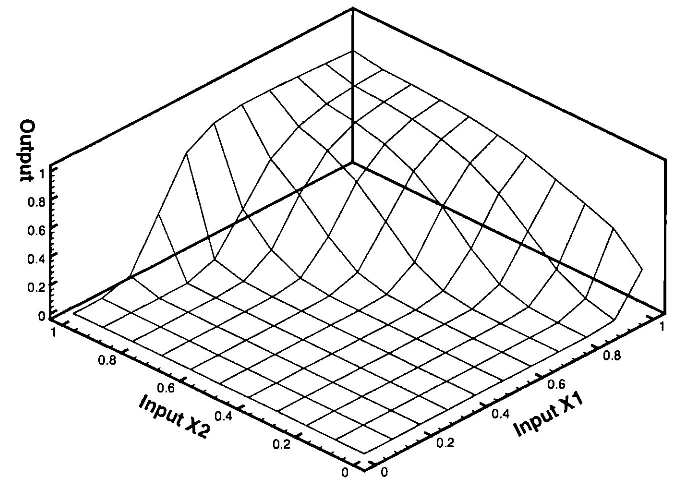


Figure 6.19c:
For an input value of +0.8 for the "Dimensional" defect.

Figure 6.19: Output Surfaces generated by the Lagrange Interpolation Polynomial Regression Network for a cause "Kind and area of die venting".



X1: Belief value in the occurrence of "Porosity" defect.

X2: Belief value in the occurrence of defect "Mismake" defect.

Output: Belief value in the occurrence of "Kind and area of die venting".

Figure 6.20a:
For an input value of 0.1 for the "Dimensional" defect.

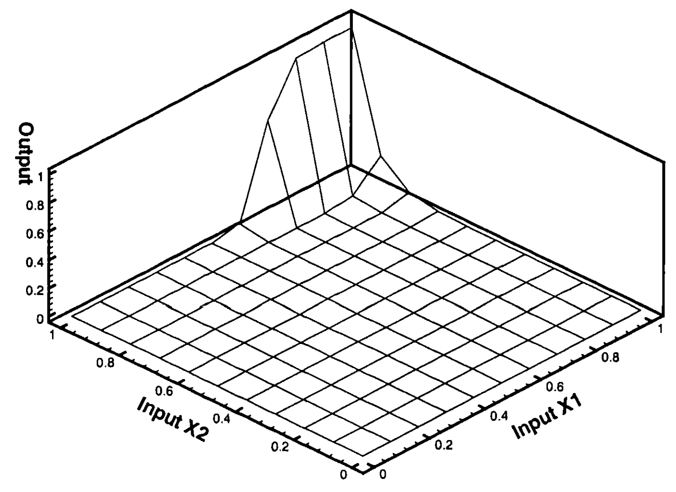


Figure 6.20b:
For an input value of 0.5 for the "Dimensional" defect.

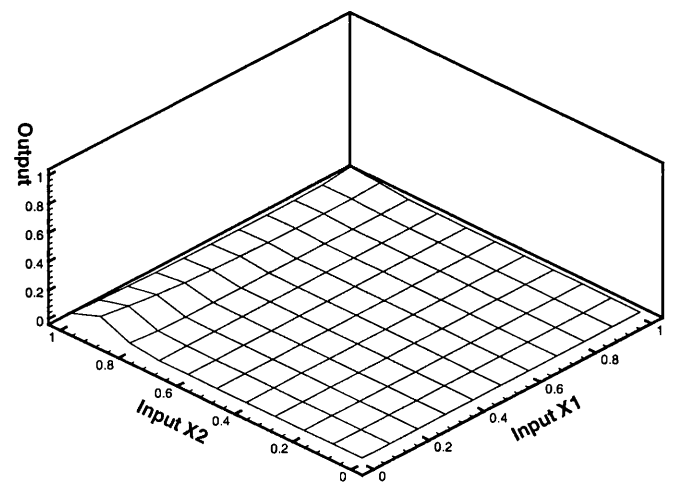
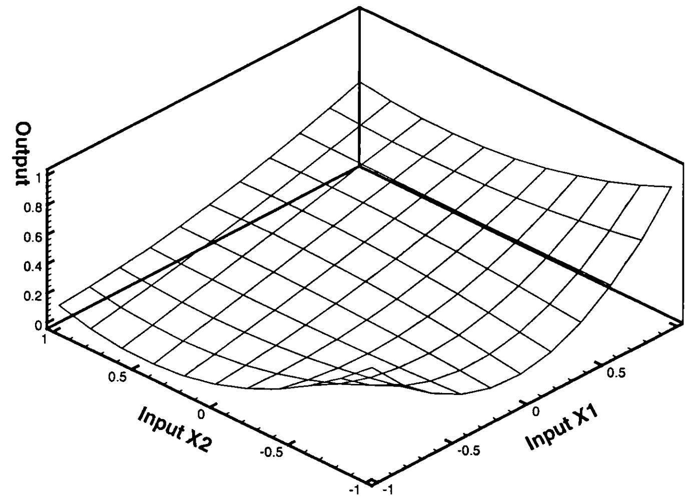


Figure 6.20c:
For an input value of 0.9 for the "Dimensional" defect.

Figure 6.20: Output Surfaces generated by the traditional neural network for a cause "Kind and area of die venting".



X1: Belief value in the occurrence of "Porosity" defect.

X2: Belief value in the occurrence of defect "Mismake" defect.

Output: Belief value in the occurrence of "The position of overflow".

Figure 6.21a:
For an input value of –0.8 for the "Dimensional" defect.

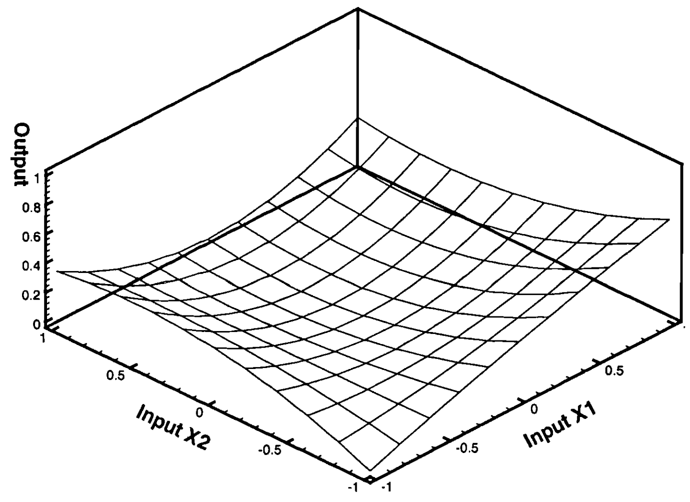


Figure 6.21b:
For an input value of 0 for the "Dimensional" defect.

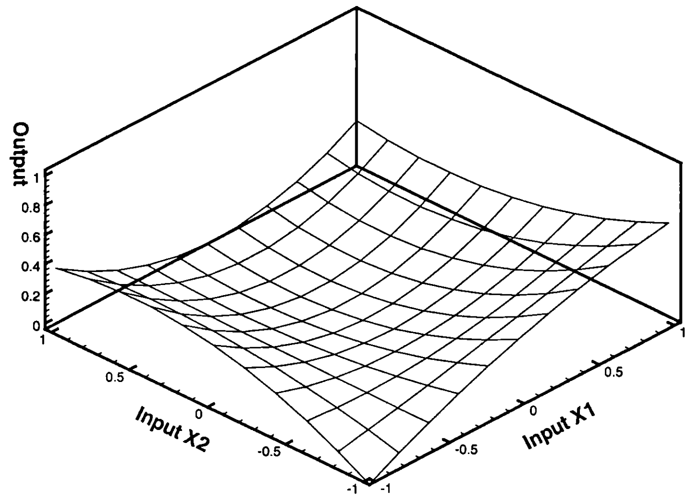
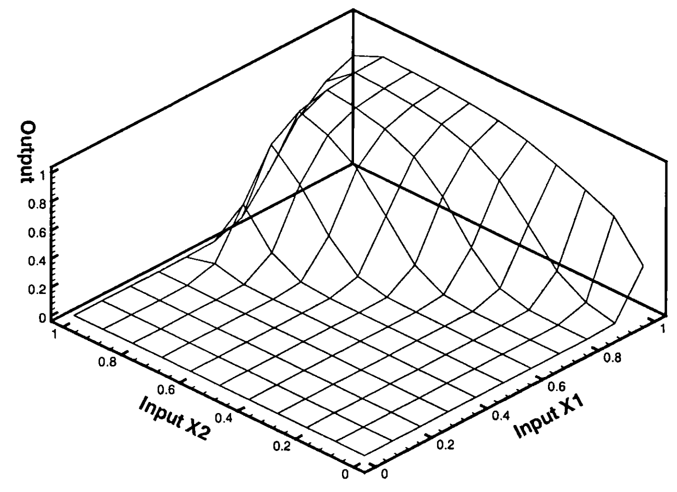


Figure 6.21c:
For an input value of +0.8 for the "Dimensional" defect.

Figure 6.21: Output Surfaces generated by the Lagrange Interpolation Polynomial Regression Network for a cause "The position of overflow".



X1: Belief value in the occurrence of "Porosity" defect.

X2: Belief value in the occurrence of defect "Mismake" defect.

Output: Belief value in the occurrence of "The position of overflow".

Figure 6.22a:
For an input value of 0.1 for the "Dimensional" defect.

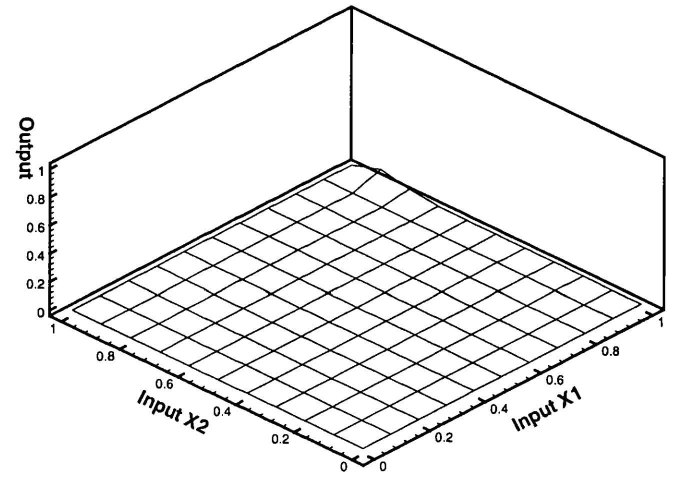


Figure 6.22b:
For an input value of 0.5 for the "Dimensional" defect.

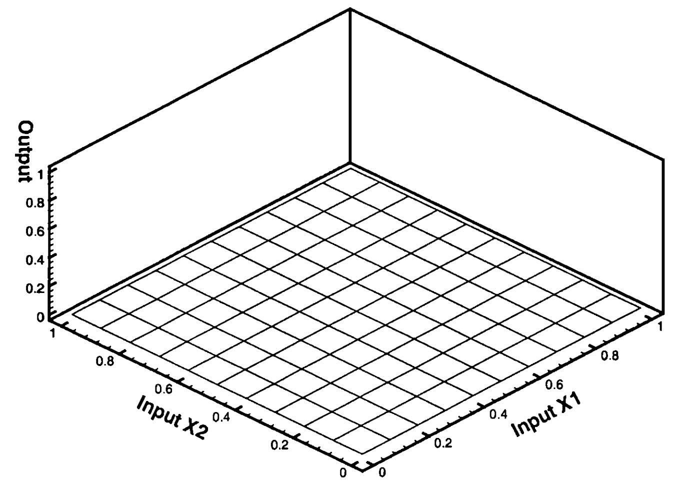


Figure 6.22c:
For an input value of 0.9 for the "Dimensional" defect.

Figure 6.22: Output Surfaces generated by the traditional neural network for a cause "The position of overflow".

6.7 Conclusion

An efficient algorithm has been presented in this chapter which retains the advantages of neural networks and overcomes their limitations in learning the input-output mapping function in the presence of noisy, limited and sparse data. It has also been shown that the algorithm has superior extrapolation abilities as compared to the multi layer neural network. The extrapolation ability was enhanced by the networks ability to constrain linear, quadratic or cubic relationships, which relates the belief values in the effects to the belief values in the causes. The dependence of the secondary weight values on the primary weight values reduced the number of unknowns to an acceptable number. Codes for both, the Lagrange Interpolation Polynomial Regression Network and a multi layer neural network, have been rewritten by the author in Matlab. Sample calculations of the proposed algorithm have been compared with the multi layer neural network for three representative cases and a real data set using the Quasi-Newton optimization algorithm.

REFERENCES

[1] R.J. Freund and W.J. Wilson, “Regression Analysis: Statistical Modelling of a Response Variable”, Academic Press, 1998 (ISBN: 0-12-267475-8).

[2] C.M. Bishop, “Neural Networks for Pattern Recognition”, Oxford University Press, 1995. (ISBN:0-19-853864-2).

[3] ftp://ftp.sas.com/pub/neural/FAQ.html

[4] J.Kalkkuhl, K.J. Hunt, and H. Fritz, “FEM-Based Neural-Network approach to non-linear modelling with application to Longitudinal vehicle dynamics control”, IEEE Transaction on Neural Network, vol 10, no 4, 1999.

[5] Ransing R.S., “An Approach for the Causal Analysis in Casting Processes based on Probabilistic Analysis, Neural Network and Design Optimisation”, *PhD Thesis, Institute of Numerical Methods in Engineering, Department of Civil Engineering, University of Wales Swansea,* Chapter 4, March 1996.

[6] Ransing R. S. and Lewis R. W., “A Semantically Constrained Neural Network for Manufacturing Diagnosis”, *International Journal of Production Research,* 1997.

[7] Matlab’s Neural Network Toolbox Website: http://www.mathworks.com/access/helpdesk/help/toolbox/nnet//nnet.shtml?BB=1

[8] Matlab's Optimisation Toolbox Website:
http://www.mathworks.com/access/helpdesk/help/pdf_doc/optim/optim_tb.pdf

Chapter 7

Conclusions and Future Work

This chapter summarises the conclusions made for the diagnostic problem researched in this thesis. Recommendations are made on how to continue this research work.

The work presented in this thesis is a part of the ongoing research on the analysis of cause and effect relationships. The "Learning from examples" ability of neural networks has been studied in detail and the issues involved in quantifying cause and effect relationships have been discussed and remedies have been recommended. The aspects researched and refined were:

- Revisit the diagnostic problem in terms of relating belief values in the occurrence of the effects and causes.

- Choice of a neural network based approach for automatically quantifying the cause and effect relationships by generating a belief value in the occurrence of a cause (or a combination of causes), given a belief value in the associated effect (or a combination of effects).

- Analyse the effect of variation of the gain value on the learning efficiency.

- Recognise the limitations to generate 'ideal' training data sets and study its influence on the neural network modelling.

- Propose a novel algorithm to associate belief values in the causes and effects by overcoming the limitations of neural networks.

With regards to quantifying the cause and effect relationship, a traditional rule based expert system approach typically requires knowledge of the degree of influence of the related cause on the occurrence of each defect. In the authors' experience, generating such a probability distribution for the entire relationship is extremely difficult if not impossible. The neural network approach offers a convenient computational tool, which, unlike the rule based approach, can adapt, learn from past examples and may then be used to quantify highly complex and inter-linked relationships. Therefore, a neural network approach was explored for analysing and quantifying cause and effect relationships.

The basic principles of neural networks, and the influence of network parameters, in conjunction with the back-propagation training algorithm for feed forward neural networks have been studied in detail. This study has shown that the number of hidden layers and also hidden nodes in each layer determines the non-linearity of a mapping function in the input and output space, which in turn determines the generalisation ability of the neural network. In a 'feed forward' algorithm, the slope of the activation function which determines the range over which output values are changed from zero to one for a given set of weighted inputs, is directly influenced by a parameter referred to as 'gain'. The influence of the variation of 'gain' on the learning ability of a neural network has been analysed. It was observed that the different values of gain used during the training process might not really achieve the desired positive control on all output values. In fact, it is shown that the influence of variation in the gain value is similar to the influence of variation in the learning rate value. A relationship between the gain, the learning rate and the initial weights of a network has also been identified. Sample examples and calculations are shown, which support this relationship. The influence of adaptive gain on the performance of the network has also been analysed. The adaptive gain procedure is advantageous because it is easy to introduce into a back-propagation algorithm. Adaptive gain has a positive effect in the learning process by modifying the magnitude, and not the direction, of the weight change. This greatly increases the learning speed by amplifying the directions in the weight space that are successfully chosen by the Gradient - Descent algorithm. A *coupled algorithm* has also been proposed in this thesis for the efficient calculation of the adaptive gain value in sequential as well as batch learning modes.

One of the major limitations of using neural network techniques is that any known information about the cause and effect relationship cannot effectively be stored within the network. As a result, when the training data set is noisy, limited, sparse and may have possible contradicting values, the network performance becomes poor. Also, in many real situations very few good quality training examples are generally available. As a result, in order to achieve the aims of this research it was necessary to store any known information about the cause – effect relationship within the network without altering the "learning from examples" ability of the network.

Neural network techniques and regression analysis methods have been discussed briefly, and their advantages and limitations for analysing cause and effect relationships have been given. Regression analysis imposes a functional form on the data and as a result it has better extrapolation abilities as compared to neural network techniques. On the contrary, neural networks extract the functional form from the data and hence, are useful when we do not know the underlying relationship between the independent and dependent variables. Also, the network architecture for neural networks is not unique, as the number of hidden units is decided by trial and error, which is generally time consuming. For regression analysis, the number of unknowns increases exponentially with the dimensionality of the input space and this is a serious limitation. As a result of this, large training data becomes necessary. For analysing cause and effect relationships, it is generally difficult to get good quality training data. Therefore, regression analysis could not be used for solving this diagnostic problem.

It was also discovered during this research that the belief variation in the occurrence of a cause, with respect to a change in the belief value of the occurrence of an effect, follows a pattern. Such a variation is generally linear, quadratic or cubic and certainly not an arbitrary higher ordered polynomial. For the first time, the use of lower ordered, one-dimensional Lagrange Interpolation Polynomials has been proposed to construct the multi dimensional hyper surfaces. A number of equidistant reference points were chosen in the input space created by belief values representing the strength of the effects. A Lagrange Interpolation polynomial and a weight value is associated with each of the said reference points. A weight value at a reference point is considered to be representative of the belief value in the cause. The reference points have been divided into two categories, referred to as primary and secondary reference points. Weight values associated with these primary reference points have been considered as independent variables (primary weight values) and other weight values associated with secondary reference points (secondary weight values), have been considered to be linearly dependent on one or more primary weight values.

It has also been shown that the proposed algorithm has superior extrapolation abilities as compared to the multi layer neural network. The extrapolation ability was enhanced by the networks ability to constrain the shape of the resulting multi-dimensional hyper surface to the known variation in the belief values in causes and effects. Sample calculations of the proposed algorithm have been compared with the neural network for four representative cases.

7.1 Future Work

This type of research is a never ending process. Although, substantial academic advancement has been achieved during the work presented in this thesis, immediate further work is outlined here:

- Generating a training data set is a time consuming task. The observations made in this thesis are based on the past training data gathered from foundries and the discussions with foundry experts along with the expertise available within the research group. The immediate future task is to collect the data, with the support of Rolls Royce Plc, on the manufacture of shells for an investment casting process and apply the algorithm developed in this thesis.

- Future research is necessary on linking primary weights with secondary weight values.

- The research needs to be extended to analyse general cause and effect relationships including the medical domain.

- The research codes need to be transformed into a user friendly software.

- Further research is required on the neural network side to study the influence of an adaptive gain value coupled with an adaptive learning rate using various optimisation methods.