AUTHOR: VISWANATHAN, A.

TITLE: Using orthogonal arrays to train artificial neural networks.

YEAR: 2005

OpenAIR citation: VISWANATHAN, A. 2005. Using orthogonal arrays to train artificial neural networks. Robert Gordon University, PhD thesis. Held on OpenAIR [online]. Available from: https://openair.rgu.ac.uk/

This work was submitted to- and approved by Robert Gordon University in partial fulfilment of the following degree:
Master of Philosophy, School of Engineering

Using Orthogonal Arrays to Train Artificial Neural Networks

A thesis submitted to
The Robert Gordon University
in partial fulfilment of the requirements for
the degree of Master of Philosophy

Alagappan Viswanathan

School of Engineering
The Robert Gordon University
Aberdeen, UK

October 2005

Dedicated to

My Wife and Daughter

with Love and Affection

Declaration

I hereby declare that this thesis is a record of work undertaken by myself. That it has not been the subject of any previous application for a degree and that all sources of information have been duly acknowledged.

Alagappan Viswanathan
2005

Abstract

The thesis outlines the use of Orthogonal Arrays for the training of Artificial Neural Networks. Such arrays are popularly used in system optimisation and are known as Taguchi Methods. The chief advantage of the method is that the network can learn quickly. Fast training methods may be used in certain Control Systems and it has been suggested that they could find application in ‘disaster control,’ where a potentially dangerous system (for example, suffering a mechanical failure) needs to be controlled quickly.

Previous work on the methods has shown that they suffer problems when used with multi-layer networks. The thesis discusses the reasons for these problems and reports on several successful techniques for overcoming them. These techniques are based on the consideration of the neuron, rather then the individual weight, as a factor to be optimised. The applications of technique and further work are also discussed.

Acknowledgements

I am indebted to my supervisor Mr Grant M Maxwell for his help, support and encouragement throughout the project. I am also indebted to my second supervisor Dr Christopher MacLeod for his valuable guidance, motivation and support throughout the research. Without their support the project would have been much more difficult and challenging.

My special note of thanks is due to Mrs Ann Reddipogu for her constant encouragement and motivation throughout the project.

I would like to thank my research group members Seth and Ananda for their support. I also would like to extend my appreciation to all the members in the School of Engineering who helped me in some way during the project.

Table of Contents

Chapter 1
Introduction

1.1 Introduction to Chapter

This chapter starts by describing the problems addressed by the project. The aims and objectives of the research are outlined and novel ideas discovered during the research are listed. A breakdown, chapter by chapter, of the thesis and a list of abbreviations used are also included.

1.2 The Nature of the Problem

The quest for Artificial Intelligence (AI) is one of the most exciting challenges that mankind has ever undertaken. The real promise of AI research is to study intelligent behaviour in humans and attempt to engineer such behaviour in a computer or other machine. Biologically inspired Artificial Neural Networks (ANNs) are one of the tools used to achieve this.

At the present time, most research into ANNs is aimed at engineering applications. Examples of such applications include Pattern Recognition, Control Systems and Signal Processing. These usually involve fairly small networks with fixed topologies, unit functionality and training methods. This has led to the adoption of popular and simple "off the shelf" networks such as Back Propagation (BP) trained Multilayer Perceptrons, Radial Basis Networks and others. An important requirement for such applications is network training time. Particularly for online Control Systems, network learning rate may be very critical.

This work presents a number of innovative methods to train neural networks using Taguchi Methods. The principle advantage of these techniques is their speed. The methods allow the weights of the network to be set quickly without having to go through a lengthy training routine. Taguchi methods allow the weights to be chosen using a pre-defined pattern of known experiments based on Orthogonal Arrays. Different levels are set for the weights in these experiments in order to find out the best combination of weights by calculation. A detailed explanation of the technique is given in Chapter 4.

1.3 Aims & Objectives

The aim of the research was to develop new training methods to train ANNs using Taguchi Methods. These networks are fairly small and have a fixed topology.

To accomplish the task the following objectives were set out at the beginning of the project.

Background Reading and Appropriate Directed Study

Appropriate directed studies were undertaken at the beginning of the research. These included attending lectures in the field of study, understanding and reproducing the work done by (MacLeod et al 1999) and understanding the work done by (Stoica et al 1997) on Neural Learning using Orthogonal Arrays.

Undertake a study of relevant literature

A literature search into the training methods of ANNs was undertaken. The search concentrated on fast training methods and other work using Taguchi Methods or Orthogonal Arrays.

Train a simple neuron using Taguchi Methods

The primary aim here was to investigate the use of Taguchi methods to train a simple (single) neuron, then progress to single layer networks, and finally to multi-layer networks.

Investigate new methods of training ANNs using Taguchi Methods

Initial experiments were concerned with finding out whether it is possible to use Taguchi Methods for ANNs training. However, when they were tried on multi-layer networks, the training failed due to interaction between the interlayer weights. A number of alternate training strategies were considered to overcome the interaction problem. These were layer-by-layer training, coding the state of neuron training method and neuron-by-neuron training. Taguchi Methods were also applied to Polynomial (power series) neurons (Maxwell et al 2002) for ANNs training. A detailed explanation of the techniques is given in Chapters 4, 5 and 6.

Investigate use of custom-made OAs for training ANNs

Under some circumstances it may be difficult to select a suitable OA table for a given problem. The use of custom-made OA tables in these situations is investigated.

Use the new methods developed to learn non-linear functions

The capabilities and limitations of the new methods are explored by applying them to learning non-linear functions.

Comparison with Previously Published Results from other Researchers

The results obtained were compared with previously published results to assess the advantages and disadvantages of the technique.

As it can be seen, all the objectives mentioned in this section have been met.

1.4 Novel Aspects of this Research

Although researchers have tried to use Taguchi Method to train the neural networks before, there are several unique aspects to the approach presented here. The most important of these are listed below.

- An innovative ‘Neuron-by-Neuron Training’, which is based on Taguchi Methods, was developed to train Artificial Neural Networks. This method enables the weights to be set very quickly without going through the lengthy training routine. The problem of interlayer weights interaction is avoided.
- Experimental results show that just after the first iteration (one pass) of training, the error reduces dramatically.
- A unique method of ‘Coding the State of Neuron Training Method’, which is based on Taguchi Methods, was also developed. This method uses

custom-made Orthogonal Array tables. This helps to select the suitable OA table to accommodate network weights.

- When compared with traditional algorithms like Back-Propagation, this method reduces the network training time significantly. It was also demonstrated in learning some non-linear functions.
- To train Polynomial (power series) neurons, another unique training method was developed using Taguchi Methods. In this method, one can set the first order weights initially, then second order weights (squared terms), and then third order weights (cubed terms) etc. The network error reduces with each increasing input power as the approximation becomes more accurate.

1.5 Thesis structure

Given below is an overview of each chapter.

Chapter 2. Literature Review

This chapter gives a review of other important work, related to the research.

Chapter 3. Introduction to Taguchi Methods

This chapter introduces the basics of Taguchi Methods, explaining the operation of the method and outlines the theory behind it. It describes orthogonal arrays and their significance in finding the best combination of factors with the minimum number of experiments.

Chapter 4. Artificial Neural Network Training using Taguchi Methods

This chapter explains the operation of Taguchi methods training by applying it to the problem of finding the weights for a simple neuron, single layer network and multi-layer networks for pattern recognition problems.

Chapter 5. New Training Methods using Taguchi Methods for ANN Training

In this chapter, the new training techniques developed to train multi-layer networks are discussed. The results obtained with different training methods developed are presented in this chapter. The advantages and disadvantages of each method are discussed.

Chapter 6. New Training Method using Custom-made OAs for ANN Training
This chapter presents a new method of training using custom-made OA tables. Results are presented and advantages and disadvantages of the method are discussed.

Chapter 7. Using New Methods in Learning Non-linear Functions
This chapter compares the results obtained using the new methods with the Back-Propagation algorithm. It was also demonstrated on non-linear training functions - sigmoid, reverse sigmoid and Gaussian (bell curve). Comparison is made between theoretical and actual outputs of the function.

Chapter 8.Conclusions
The final chapter revisits the objectives outlined in the first chapter and critically assesses the success of the project. It also suggests some further work.

Published papers and some further results are included in appendices.

1.6 Abbreviations used in text

Artificial Intelligence - AI

Artificial Neural Networks - ANN

Neural Network – NN

Taguchi Methods – TM

Orthogonal Arrays - OA

Back Propagation - BP

Chapter 2

Literature Review

2.1 Introduction to Chapter

This chapter reviews the previous work done related to the research topic. The literature review is primarily focused on the main research area of using Taguchi Methods or Orthogonal Arrays for ANN training. A number of papers proposed the use of Taguchi Methods and Orthogonal Arrays for neural network design unrelated to training: for example, choosing the number of neurons in the layer, training sampling selection, etc. These are also discussed briefly in this chapter. To make a comparison with the present work and to gain a better understanding of its capabilities, other fast training methods for training ANNs are also considered.

2.2 Taguchi Methods and Orthogonal Arrays for ANN training

The idea of using Taguchi Methods and Orthogonal Arrays to train Artificial Neural Networks was originated by C. MacLeod in 1994 (MacLeod et al 1999) at The Robert Gordon University and implemented by his student (Dror 1995) in an MSc project. Another group at the JPL research centre in NASA also developed the same idea independently at around the same time (Stoica et al 1997).

The initial work by (MacLeod et al 1999) was demonstrated on a simple character recognition problem. It was shown that Taguchi Methods offered the potential to train feed-forward neural networks and had advantages in terms of their speed and unsupervised training properties. In some circumstances, it proved faster than other algorithms and in general character recognition problems, it was up to 10 times faster than the back-propagation algorithm.

At the JPL research centre in NASA (Stoica et al 1997), the method was tested on a multi-layer neural network. It was shown that the method could be used iteratively to reduce the error further over a number of runs. After each iteration, the search interval was shrunk to narrow down the search space. To improve the final result, a local search (e.g. gradient-based) was suggested. Further work was also suggested to obtain an appropriate OA table size for a given network.

The general capabilities and limitations of Taguchi Methods for ANN training were explored by (Maxwell et al 2002) and demonstrated on some more pattern recognition problems. The issue of obtaining suitable OA tables was highlighted and it was suggested that custom-made tables could be generated to accommodate the network weights. The other problem discussed was the interaction between interlayer weights. As discussed elsewhere in the thesis, if the method is applied to a multi-layer network, the resultant network will often work poorly, due to the problem of interaction between the weights, since the conventional method can only handle a small amount of interaction between the factors. It was suggested that one possible way around this problem was to train one layer at a time (randomising the weights in the other layer). To establish whether this was a reliable method of training, further work was suggested.

The method was tested on a simple non-linear function and it was highlighted that it offers potential for neural control applications, because of the training speed and the fact that the networks involved are usually small.

It was suggested that power series (polynomial) neurons are more suitable for the Taguchi Method of training because it allows the setting of first order weights initially, then, independently, second order weights then the third order weights and so on. The network error reduces with each increasing power.

Previous experimental work (Maxwell et al 2002) had shown that the method could successfully train single layer networks. However, interaction between layers precluded the successful reliable training of multi-layer networks. A paper, based on the current research work, by (Viswanathan et al 2005) describes a number of successful strategies, which can be used to overcome this problem. Successful results were produced using these methods and it was also shown that the new methods could be used to map non-linear functions. This paper is included in Appendix A1.

2.3 Taguchi Methods and Orthogonal Arrays for general ANNs design

Quite a number of papers have also proposed the use of Taguchi Methods or Orthogonal Arrays for optimum neural network design. For example, choosing the number of neurons in a layer, training selection, etc. Some of these can lead indirectly to better training speeds or faster convergence. Some are aimed at a specific application. To give the reader some idea of the range of these papers, some are discussed in this section.

These developments are parallel to the field, but not the same as the current research work.

(Young-Sang Kim et al 2004) proposed the robust design of multi-layer feed-forward ANNs. ANNs have been successfully used for solving a wide variety of problems; however, determining a suitable set of structural and learning parameter values still remains a difficult task. This paper is concerned with the robust design of multi-layer feed-forward ANNs, trained by the BP algorithm and develops a systematic experimental strategy which emphasises simultaneous optimisation of BP neural network parameters under various noise conditions. The problem is formulated as a Taguchi dynamic parameter design, together with a fine-tuning of the BP neural network output.

(Ming-Der Jean et al 2005) proposed the application of an artificial neural network with a Taguchi orthogonal experiment to develop a robust and efficient method of depositing alloys with a favourable surface morphology by a specific microwelding hardfacing process. An ANN model performs self-learning by updating weights and repeating learning epochs. The network can be constructed based on the data obtained from experiments. The root of mean squares (RMS) error can be minimized by applying results obtained from training and testing samples, such that the predicted and experimental values exhibit a good linear relationship. The experimental results reveal that the coating is greatly improved by optimising the coating conditions and is accurately predicted by the neural network model. The combination of the network model with Taguchi-based experiments is demonstrated as an effective and intelligent method for developing a robust, efficient, high-quality coating process.

(Yang et al 1999) proposed a method of neural network design using Taguchi Methods. One of the major difficulties in ANN applications is the selection of the network configuration parameters and of learning algorithm coefficients for fast convergence. This paper develops a network design by combining the Taguchi Method and BP training with an adaptive learning rate for minimum training time. Analysis and experiments show that the optimal design parameters can be determined in a systematic way, thereby avoiding a lengthy trial-and-error process.

Optimal design of ANNs using Taguchi Methods was suggested by (Khaw et al 1995). The design of ANN involves the selection of an optimal set of design parameters to achieve fast convergence during training and the required accuracy during recall. This paper describes an innovative application of the Taguchi Method for the determination of these parameters to meet the training speed and accuracy requirements. The feasibility of using this approach is demonstrated in this paper by optimising the design parameters of a BP neural network for determining operational policies for a manufacturing system.

(Packianather et al 2000) proposed optimising the parameters of multilayered feed forward ANN through Taguchi Design. Being a parallel approach, the method offers considerable benefits in time and accuracy when compared with the conventional serial approach of trial and error. The use of Taguchi Methods ensures that the quality of the ANN is taken into account at the design stage.

(Mo-Chung Lee et al 1999) also proposed the use of Taguchi Methods to develop the structure of a BP neural network. The method makes it easy to determine the optimal number of hidden nodes, learning rate and momentum term.

Using Taguchi Methods to control errors in Multi-Layered Perceptrons is proposed by (Peterson et al 1995). Network errors can occur when the training data does not faithfully represent the required function due to noise or low sampling rates. The paper reports several experiments whose purpose was to rank the relative significance of these error sources and thereby find ANN design principles for limiting the magnitude and variance of network errors.

(Rowlands et al 1996) explains how optimal design can be achieved by using design of experiments in conjunction with neural networks. It is common practice in industry to find the optimal design through the Taguchi methods. In order to identify the optimal design, Taguchi methods replace the need for running a full factorial design of experiments by a fractional factorial design using orthogonal arrays. However, the compromise between the use of fractional factorial design and full factorial design requires some assumptions to be made in identifying the optimal design parameters and consequently leads to some uncertainty in the result. The neural network was trained using the results of a fractional factorial design for an intelligent sensor example. The neural network was then used to predict the response values for the full factorial design. A comparison between the Taguchi method and the neural network approach highlights the superior results produced by the neural network.

(Su, Chao-Ton et al 2000) proposes an approach using a neural network and simulated annealing for parameter design optimisation. Parameter design optimisation has extensive industrial applications, including product development, process design and operational condition setting. The parameter design optimisation problems are complex because non-linear relationships and interactions may occur among parameters. To resolve such problems, engineers commonly employ the Taguchi method. However, the Taguchi method has some limitations in practice. In this work, the authors present a means of improving the effectiveness of the optimisation of parameter design. The proposed approach employs the neural network and simulated annealing, and consists of two phases. Phase 1 formulates an objective function for a problem using a neural network method to predict the value of the response for a given parameter setting. Phase 2 applies the simulated annealing algorithm to search for the optimal parameter combination. A numerical example demonstrates the effectiveness of the proposed approach.

(Chang et al 2002) proposed the selection of training samples for model updating using ANNs. One unique feature of ANNs is that they have to be trained to function. In developing an iterative ANN technique for model updating structures, it was shown that the number of training samples required increases exponentially as the number of parameters to be updated increases. Training the neural networks using these samples becomes a time-consuming task. In this study, the authors investigate the use of OAs for the sample selection. The results indicate that the OA method can significantly reduce the number of training samples without affecting too much the accuracy of the ANN prediction.

2.4 Other methods to improve ANNs training

A number of papers have proposed different training algorithms and other methods to improve the network training speed. These involve developing a new algorithm or improving the existing algorithms like BP, etc. It is not possible to cover all of these papers here due to the large number of them; however, some of them are reviewed in this section to gain better understanding.

Although these papers discuss several methods of improving ANN training, the methods developed in this research work are completely different from the other methods and as such, are unique.

A new fast high-order neural network learning algorithm for pattern recognition was proposed by (Zhang et al 2004). The new algorithm uses the properties of trigonometry to reduce and control the number of weights of a third-order network used for invariant pattern recognition.

A fast training algorithm which could be used instead of BP was suggested by (Yamada et al 1994), which is based on the Newton-Rapson method. It is well known that BP as a gradient-descent algorithm can get stuck in local minima and this method is designed to avoid that problem. Comparison is made with back-propagation algorithm.

An improved neural network learning algorithm was proposed by (Altun et al 1997). Using this technique, an improvement on the BP algorithm is obtained. The technique is based on manipulating the input data so that the redistribution of the input domain results in quick learning. The results presented in this paper show an acceleration by a factor of 8.7 over standard BP.

(Dubrovin et al 2002) suggested another quick method of neural network training. An algorithm for training neural networks, which increases the convergence of network training, is developed. The results of experiments in practical problem solving on the basis of the proposed algorithm are shown.

Some learning algorithms for neurocomputing have a high computational complexity that makes them inefficient. (Looney et al 1992) investigate a new approach for learning that is quick relative to BP. The key feature of this approach is that it uses a different synaptic weight set for each class of objects that is learned.

Another algorithm was proposed by (Kawata et al 1998) to overcome the problems posed by BP (like low rate of convergence, low recognition rate for unlearned data, initial random weights and one point search, etc). The proposed method uses a genetic algorithm, which searches several search points at the same time for improvement dependent on the initial random and limited part solution.

The algorithms reviewed in this section are included as alternative fast training methods. They offer other techniques which can result in improvement in learning. However, it should be noted that it is not possible to compare the method reported here with them directly as the papers often do not contain actual data on algorithm speed, and when they do it is based on data which is not directly comparable to the experiments illustrated in this thesis.

Chapter 3
Introduction to Taguchi Methods

3.1 Background & Overview of Taguchi Methods

After the Second World War, the allied forces found that the quality of the Japanese telephone system was extremely poor and totally unsuitable for long-term communication purposes. To improve the system the allied command recommended establishing research facilities in order to develop a state-of-the-art communication system. The Japanese founded the Electrical Communication Laboratories (ECL) with Dr. Genichi Taguchi in charge of improving the R&D productivity and enhancing product quality. He observed that a great deal of time and money was expended on engineering experimentation and testing (Ranjit 1990). Little emphasis was given to the process of creative brainstorming to minimise the expenditure of resources. He noticed that poor quality cannot be improved by the process of inspection, screening and salvaging. No amount of inspection can put quality back into the product. Therefore, he believed that quality concepts should be based upon, and developed around, the philosophy of prevention.

Taguchi started to develop new methods to optimise the process of engineering experimentation. He believed that the best way to improve quality was to design and build it into the product. He developed the techniques which are now known as *Taguchi Methods*. His main contribution lies not in the mathematical formulation of the design of experiments, but rather in the accompanying philosophy. His concepts produced a unique and powerful quality improvement technique that differs from traditional practices. He developed manufacturing systems that were “robust” or insensitive to daily and seasonal variations of environment, machine wear and other external factors.

His philosophy had far reaching consequences, yet it is founded on three very simple concepts. His techniques arise entirely out of these three ideas.

The concepts are:

1. Quality should be designed into the product and not inspected into it.
2. Quality is better achieved by minimising the deviation from a target. The product should be so designed that it is immune to uncontrollable environmental factors.
3. The cost quality should be measured as a function of deviation from the standard and the losses should be measured system-wide.

Taguchi viewed quality improvement as an ongoing effort. He continually strived to reduce the variation around the target value. The first step towards improving quality is to achieve the population distribution as close to the target value as possible. To accomplish this, Taguchi designed experiments using especially constructed tables known as "Orthogonal Arrays" (OA). The use of these tables makes the design of experiments very easy and consistent.

The Taguchi Method is applied in four steps.

1. Brainstorm the quality characteristics and design parameters important to the product/process.
2. Design and conduct the experiments.
3. Analyse the results to determine the optimum conditions.
4. Run a confirmatory test using the optimum conditions.

Taguchi methods start with an assumption that we are designing an engineering system - either a machine to perform some intended function, or a production process to manufacture some product or item. Since we are knowledgeable enough to be designing the system in the first place, we generally will have some understanding of the fundamental processes inherent in that system. Basically, we use this knowledge to make our experiments more efficient. We can skip all the extra effort that might have gone in to investigating interactions that we know do not exist. Without going into the details, it has been shown that this can decrease the level of effort by a factor of ten or twenty and sometimes much more.

Another distinction of Taguchi methods is the recognition that there are variables that are under our control and variables that are not under our control. In Taguchi terms, these are called Control Factors and Noise Factors, respectively.

This chapter gives a general introduction to Taguchi Methods. A detailed analysis of results using the method is beyond the scope of the thesis. Hence, we will limit the technique's applicability to the main research topic.

3.2 An Insight into Orthogonal Arrays (OA) & Taguchi Methods

The technique of laying out the conditions (designs) of experiments involving multiple factors was first proposed by Sir R. A. Fisher, in the 1920s (Ranjit 1990). The method is popularly known as *factorial design* of experiments. A full factorial design identifies all possible combinations for a given set of factors. Since most industrial experiments involve a significant number of factors, a full factorial design results may involve a large number of experiments.

Factors are the different variables which determines the functionality or performance of a product or system. Factors are:

- design parameters that influence the performance.
- input that can be controlled.
- included in the study for the purpose of determining their influence upon the most desirable performance.

In a heat treatment experiment, for example, a factor can be "cooling rate" or "temperature" etc. Each factor may be set to different levels. Hence for the same experiment the levels can be "slow cooling" and "fast cooling" or "low temperature" and "high temperature" etc. depending on the application.

For example, consider a design with three variables (factors A, B and C), each of which can be set at two different values. For convenience, these values are denoted as levels, 1 and 2. A full factorial experiment requires $2^3 = 8$ experiments, as shown in Table 3-1. On the other hand, one can get as much useful data using four experiments as indicated in Table 3-2, which is an L4 OA (general properties of OA are given in section 3.2.1 of this chapter).

Experiments	**A**	**B**	**C**
1	1	1	1
2	1	1	2
3	1	2	1
4	1	2	2
5	2	1	1
6	2	1	2
7	2	2	1
8	2	2	2

Table 3-1. Full factorial experiments table

Experiments	**A**	**B**	**C**
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Table 3-2. Orthogonal Array L4

For example, in an experiment involving seven factors, each with two levels, the total number of combinations will be 128 (2^7). To reduce the number of experiments to a practical level, only a small set from all possibilities is selected. The method of selecting a limited number of experiments which produces the most information is known as a partial factorial experiment. Although this shortcut method is well known, there are no general guidelines for its application or the analysis of the results obtained by performing the experiments (Ranjit 1990).

Taguchi's approach complements these two important areas. Taguchi constructed a special set of *Orthogonal Arrays* (OA) to lay out his experiments. By combining existing orthogonal latin squares in a unique manner, Taguchi prepared a new set of standard OAs which could be used for a number of experimental situations. He also devised a standard method for analysis of the results. A single OA may accommodate several experimental situations. Commonly used OAs are available for 2, 3 and 4 levels. The combination of standard experimental design techniques and analysis methods in the Taguchi approach produces consistency and reproducibility.

3.2.1 Properties of the OA

A common OA for 2 level factors is shown in table 3-3. This array, designated by the symbol L8, is used to design experiments involving up to seven 2 level factors. The array has 8 rows and 7 columns. Each row represents a trial condition (experiment) with factor levels indicated by the numbers in the row. The vertical columns correspond to the factors specified in the study.

Experiments	Factors						
	A	B	C	D	E	F	G
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

Table 3-3. Orthogonal Array L8 (2^7)

Assume that a variable (i.e. a design parameter under investigations) can take n different values, $v_i \ldots v_n$. Assume that a total of m experiments are conducted. Then a set of experiments is *balanced with respect to the variable* if:

(i) $m = kn$, for some integer k;

(ii) each of the values, v_i, is tested in exactly k experiments.

An experiment is *balanced* if it is balanced with respect to each variable under investigation. For example, in L8 OA shown in table 3-1, each column contains four level 1 and four level 2 conditions for the factor assigned to the column. It is easy to see that all columns provide four tests under the first level of the factor, and four tests under the second level of the factor.

The idea of balance ensures equal chance is given to each level of each variable. Similarly, we want to give equal attention to *combinations of two variables*. Assume that we have two variables, A (values: $a_i, \ldots, a_n$) and B (values $b_i, \ldots, b_m$). Then the set of experiments is orthogonal if each pair-wise combination of values, (a_i, b_j) occurs in the same number of trials.

For example, in L8 OA shown in table 3-3, two factors with 2 levels combine in four possible ways:

$$(1,1), (1,2), (2,1) \text{ and } (2,2)$$

Note that any *two* columns of an L8 OA have the same number of combinations of (1,1), (1,2), (2,1) and (2,2). This is one of the features that provide the orthogonality among all the columns (factors).

When two columns of an array form these combinations, the same number of times (two times in this case), and all columns provide the same number of tests under the first level of the factor, and the same number of tests under the second level of the factor, then the columns are said to be balanced and orthogonal. Thus, all seven columns of an L8 array are orthogonal to each other.

In Taguchi design, the array is orthogonal, which means the design is balanced so that factor levels are weighted equally. The real power in using an OA is the ability to evaluate several factors in a minimum of tests. This is considered an efficient experiment since much information is obtained from a few trials.

Consider the following array with 12 rows and 11 columns:

$$\begin{array}{ccccccccccc}
0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&1&1&0&1&0&0&0\\
0&1&1&1&0&1&1&0&1&0&0\\
0&0&1&1&1&0&1&1&0&1&0\\
0&0&0&1&1&1&0&1&1&0&1\\
1&0&0&0&1&1&1&0&1&1&0\\
0&1&0&0&0&1&1&1&0&1&1\\
1&0&1&0&0&0&1&1&1&0&1\\
1&1&0&1&0&0&0&1&1&1&0\\
0&1&1&0&1&0&0&0&1&1&1\\
1&0&1&1&0&1&0&0&0&1&1\\
1&1&0&1&1&0&1&0&0&0&1
\end{array}$$

Pick any two columns, say the first and the last.

$$\begin{array}{cc}
0&0\\
1&0\\
0&0\\
0&0\\
0&1\\
1&0\\
0&1\\
1&1\\
1&0\\
0&1\\
1&1\\
1&1
\end{array}$$

Each of the four possible rows are,

$$0\ 0, \qquad 0\ 1, \qquad 1\ 0, \qquad 1\ 1$$

And they all appear *the same number of times* (three times, in fact). That is the property makes it an orthogonal array.

Only 0's and 1's appear in that array, but for use in statistics

$$0 \qquad \text{or} \qquad 1$$

The first column might be replaced by,

"butter" or "margarine" ,

and the second column might be replaced by,

"sugar" or "no sugar" ,

and so on.

Since only 0's and 1's appear, this is a *2-level array*. There are 11 columns, which means one can vary the levels of up to *11 different variables,* and 12 rows, which means one is going to conduct 12 different experiments.

The array forces all experimenters to design identical experiments. Experimenters may select different designations for the columns but the eight trial runs will include all combinations independent of column definition. Thus the OA assures consistency of design by different experimenters (Ranjit 1990).

To design an experiment, the most suitable orthogonal array is selected. Next, factors are assigned to the appropriate columns, and finally, the combinations of the individual experiments (called the trial conditions) are described. Let us assume that there are at most seven 2 level factors in the study. Call these factors A, B, C, D, E, F and G, and assign them to columns 1, 2, 3, 4, 5, 6 and 7 respectively of an L8 array. The table identifies the eight trials needed to complete the experiment and the level of each factor for each trial run. Each experimental set up is determined by reading numerals 1 and 2 appearing in the rows of the trial runs. A full factorial experiment would require 2^7 or 128 runs, but would not provide appreciably more information.

Experimental design using OAs is attractive because of experimental efficiency. Generally speaking, OA experiments work well when there is minimal interaction among factors, i.e. the factor influences on the measured quality objectives are independent of each other and are linear - in other words, the outcome is directly proportional to the linear combination of individual factor effects. OA design identifies the optimum condition and estimates performance in this situation accurately. If, however, the factors interact with each other and influence the outcome in a non-linear manner, there is still a good chance that the optimum condition will be identified accurately (Ranjit 1990), but the estimate of performance at the optimum can be poor. The degree of inaccuracy in performance estimates will depend on the degree of complexity of interactions among all the factors.

3.2.2 Designing the Experiment

Before designing an experiment, knowledge of the product/process under investigation is of prime importance for identifying the factors likely to influence the outcome.

The aim of the analysis is primarily to seek answers to the following three questions:

1. What is the optimum condition?
2. Which factors contribute to the results and by how much?
3. What will be the expected result at the optimum condition?

Consider an example. An experimenter has identified three controllable factors for a plastic moulding process. Each factor can be applied at two levels (Table 3-4). The experimenter wants to determine the optimum combination of the levels of these factors and to know the contribution of each to product quality.

FACTORS / LEVELS	A. Injection Pressure	B. Mould temperature	C. Set Time
LEVEL 1	A_1 = 250 psi	B_1 = 150 ^oF	C_1 = 6 sec.
LEVEL 2	A_2 = 350 psi	B_2 = 200 ^oF	C_2 = 9 sec.

Table 3-4. Factors and levels for molding process

There are 3 factors, each at 2 levels, thus an OA of L4 is suitable which is shown in Table 3-5.

Experiments	FACTORS		
	A	B	C
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Table 3-5. An experiment layout using L4 OA

This configuration is a convenient way to layout a design. Since an L4 has 3 columns, 3 factors can be assigned to these columns in any order. Having assigned the factors, their levels can also be indicated in the corresponding column. There are four independent experimental conditions in an L4. These conditions are described by the numbers in the rows.

A full set of experiments for this process would require eight different experiments (full factorial design = 2^3) as opposed to the four which are needed for the Taguchi version of the experiment using L4 OA. As previously noted, the saving involved in using the Taguchi method becomes more significant as the number of levels or factor increases (Ranjit 1990).

To analyse the results, there must be a way of comparing the results produced by each experiment. In this example, one could measure the quality characteristic, Y – the lower the better, of the moulded products.

So, having undertaken the experiments and obtained the results, it is now possible to calculate the best levels to use with each factor. Let us assume, for example, the results obtained are as shown in Table 3-6.

Experiments	FACTORS			Result (quality characteristic)
	A	B	C	
Y1	1	1	1	30
Y2	1	2	2	25
Y3	2	1	2	34
Y4	2	2	1	27

Table 3-6. Results for experiments

One can now find the effect of each level in each factor by averaging the results which contain that level and that factor.

A1 = (Y1 + Y2)/2 = (30 + 25)/2 = 27.5
A2 = (Y3 + Y4)/2 = (34 + 27)/2 = 30.5
B1 = (Y1 + Y3)/2 = (30 + 34)/2 = 32.0
B2 = (Y2 + Y4)/2 = (25 + 27)/2 = 26.0
C1 = (Y1 + Y4)/2 = (30 + 27)/2 = 28.5
C2 = (Y2 + Y3)/2 = (25 + 34)/2 = 29.5

From the above we can see that the best combination of factors is A1, B2, and C1. These are the factors which produce the lowest results.

3.2.3 Designs with Interaction

The term interaction is used to describe a condition in which the influence of one factor upon the result is dependant on the condition of another. Two factors A and B are said to interact when the effect of changes in level A, determines the influence of B and vice versa.

Consider the following example. Temperature and humidity appear to have strong interaction with respect to human comfort. An increase in temperature alone may cause slight discomfort but the discomfort increases as humidity increases. Assume the comfort level is dependant only upon two factors T and H, and is measured in terms of numbers ranging from 0 to 100. If T and H are each allowed to assume levels of T1, T2, H1 and H2, a set of experimental data may be obtained and is represented by Table 3-7

	T1	T2	Total
H1	62	80	142
H2	75	73	148
Total	137	153	290

Table 3-7. Layout for Experiment with Two 2 level Factors with Interaction

The data plotted in Figure 3-1 shows an interaction between the two factors, since the lines cross each other. If the lines are parallel, it means there is no interaction. If the lines are not parallel or not crossing each other, the factors may interact, albeit weakly.

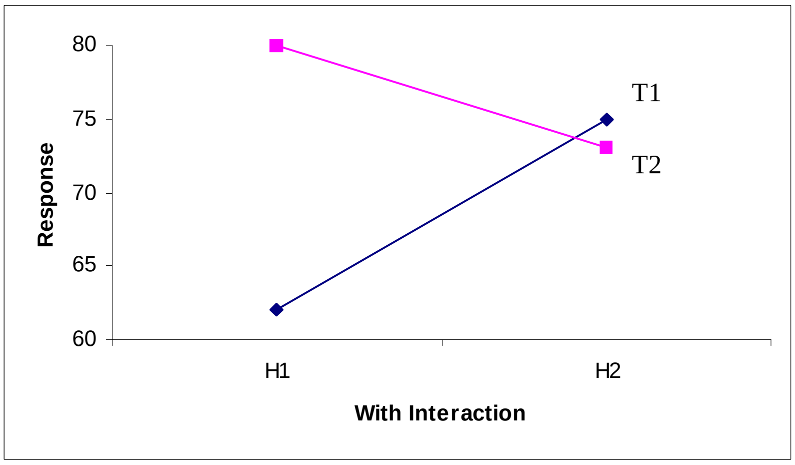


Figure 3-1. Main effects of factors T and H show Interaction

This graphical method reveals if interaction exists and may be calculated from the experimental data.

Assigning factors to columns

Experimental design using Taguchi OA's is simple and straightforward when there is no need to include interactions. It requires a little more care to design an experiment where interactions are to be included. In Taguchi OA's the effect of interactions are mixed with the main effect of a factor assigned to some other column. For example, in the L4 shown in Table 3-8 with factors A and B assigned to columns 1 and 2, interaction effects of A x B will be contained in column 3. If the interactions of A x B are of no interest, a third factor C can be assigned to column 3. The effect of interaction A x B will then be mixed with the main effect of factor C.

Experiments	A	B	A x B C
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Table 3-8. Orthogonal Array L4 with Two 2 level Factors

The following Standard Orthogonal Arrays are commonly used to design experiments:
2-Level Arrays: L4, L8, L12, L16, L32
3-Level Arrays: L9, L18, L27
4-Level Arrays: L16, L32

Some standard arrays also accommodate factors with mixed levels. In some situations, a standard OA is modified to suit a particular experiment requiring factors of mixed levels which are well explained in many texts (Ross 1988).

One of the limitations of conventional Taguchi Methods for Neural Network problems is that published Orthogonal Arrays are of fixed and often inconvenient size for the network. Very large OAs are not often published and these may be needed for larger networks. One way around these problems is to generate tables custom-made for the particular network design. The tables used in conventional Taguchi Methods are actually only a subset of those which it is possible to use. Taguchi techniques belong to a family of similar methods called "*n fractional methods*". Like the tables Taguchi chose, these are also suitable for optimisation problems and may be applied to Neural Networks in the same way. It is therefore possible to use these alternative techniques to generate tables of different sizes and structures. Details of suitable methods for generating tables from first principles may be found in, for example, (Owen 2004) and (Dey 1985) and a library of over 200 Orthogonal Arrays in (Sloane 2004).

3.2.4 Triangular table of Interaction & Linear Graphs

Each OA has a particular set of linear graphs and a triangular table associated with it. The Triangular Table of Interaction presents information about which columns interact. A triangular table therefore contains information about the interaction of the various columns of an OA. The table 3-9 should be interpreted in the following way. The number in the parenthesis at the bottom of each column identifies the column. To find in which column the interaction between columns 4 and 6 will appear, move horizontally across (4) and vertically from (6), the intersection is "2" in the tables. Thus the interaction effects between columns 4 and 6 will appear at column 2. In similar manner, other interacting columns can be identified.

Column:	1	2	3	4	5	6	7
	1	3	2	5	4	7	6
		(2)	1	6	7	4	5
			(3)	7	6	5	4
				(4)	1	2	3
					(5)	3	2
						(6)	1
							(7)

Table 3-9. Interaction between two columns in an L8 OA

This triangular table facilitates laying out experiments with interactions. The table greatly reduces the time and increases the accuracy of assigning proper columns for interaction effects.

To further enhance the efficiency of the experimental layout, Taguchi created line diagrams based on the triangular tables known as Linear Graphs. These diagrams represent standard experimental designs.

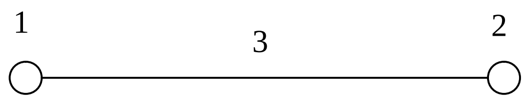


Figure 3-2. Linear graph for L4

Linear graphs are made up of numbers, dots and lines as shown in Figure 3-2, where a dot and its assigned number identifies a factor, a connecting line between two dots indicates interaction and the number assigned to the line indicates the column number in which interaction effects will be compounded. Factors 1 and 2 are assigned to columns 1 and 2 respectively and column 3 is assigned for interaction between factors 1 and 2.

In designing experiments with interactions, the triangular tables are essential; the linear graphs are complementary to the tables.

Chapter 4

Artificial Neural Network Training using Taguchi Methods

4.1 Introduction to Chapter

This chapter discusses the basic methodology developed to train the Artificial Neural Networks using Taguchi Methods. The method is first tried on a simple neuron, and then tested on multi-layer networks. The results obtained are discussed below.

4.2 Training a Simple Neuron

Initial experiments are performed with a simple, single perceptron, type neuron. Consider the four-input neuron as shown in Figure 4-1. The inputs are i_0, i_1, i_2 and i_3 and w_0, w_1, w_2 and w_3 are the corresponding weights (Wasserman 1989).

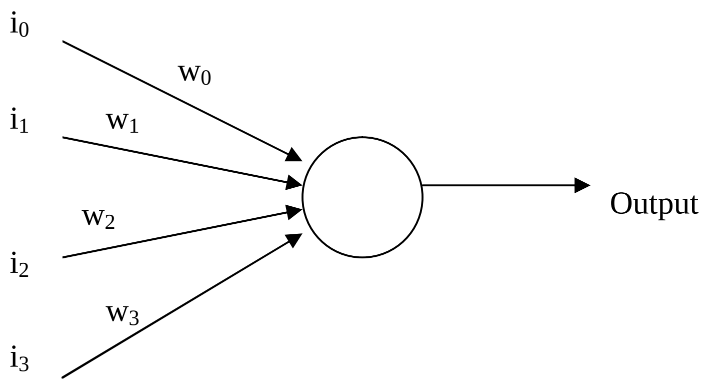


Figure 4-1. A simple neuron with inputs and weights

The sum is given by:

$$Sum = i_0 w_0 + i_1 w_1 + i_2 w_2 + i_3 w_3$$

and the output:

$$\text{Output} = \frac{1}{1+e^{-Sum}}$$

An L9 orthogonal array may be used to conduct the experiment, which is shown in Table 4-1.

Experiment No.	Column 1	Column 2	Column 3	Column 4
Exp 1	1	1	1	1
Exp 2	1	2	2	2
Exp 3	1	3	3	3
Exp 4	2	1	2	3
Exp 5	2	2	3	1
Exp 6	2	3	1	2
Exp 7	3	1	3	2
Exp 8	3	2	1	3
Exp 9	3	3	2	1

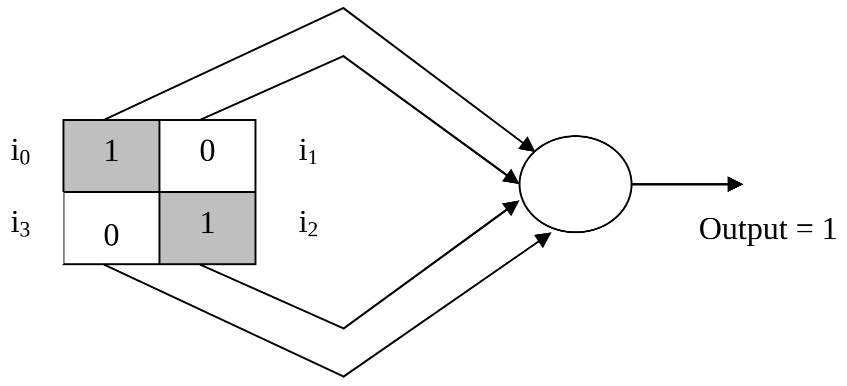
Table 4-1. L9 Orthogonal Array

The L9 OA is a three level array (levels 1, 2 and 3 as shown in Table 4-1) with 9 experiments.

4.2.1 Experimental Design

The following test parameters for two patterns are used to conduct the experiments, Figure 4-2.

Pattern 1:



Pattern 2:

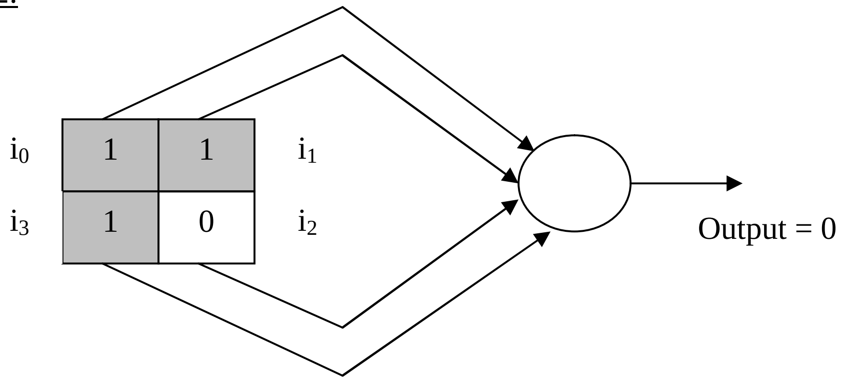


Figure 4-2. Simple neuron with two input patterns

Weights w_0, w_1, w_2 and w_3 are assigned to column 1, column 2, column 3 and column 4 of L9 OA respectively. The task of any training algorithm is to choose values for these weights, so as to produce the correct outputs from the neuron for a set of given input vectors.

To make it simple, for this example, let us say that each weight can have only one of three fixed values, L9 being a three level array. In other words, the values have been quantised which the weights may take. For example, in experiment number 1 all four weights are at value number one i.e., level 1. In experiment number 2 the first weight is at level 1 and the other three weights at level 2 etc. (refer Table 4-1). The weights corresponding to the used levels are:

Level 1 = -2

Level 2 = 0

Level 3 = 2

These levels are just used as a starting point for the weights; however, these will be fine-tuned and the level range will be reduced to get closer to the optimum weights.

Therefore, when the levels are mapped in L9 OA, the experimental layout would be as shown in Table 4-2.

Experiment No.	Weight w_0	Weight w_1	Weight w_2	Weight w_3
Exp 1	-2	-2	-2	-2
Exp 2	-2	0	0	0
Exp 3	-2	2	2	2
Exp 4	0	-2	0	2
Exp 5	0	0	2	-2
Exp 6	0	2	-2	0
Exp 7	2	-2	2	0
Exp 8	2	0	-2	2
Exp 9	2	2	0	-2

Table 4-2. L9 Orthogonal Array experimental layout

For the first pattern, for experiment no.1, the test parameters are therefore:

Inputs $i_0 = 1$

$i_1 = 0$

$i_2 = 1$

$i_3 = 0$

Weight $w_0 = -2$

$w_1 = -2$

$w_2 = -2$

$w_3 = -2$

Output = 1

The sum and sigmoid output are calculated. Then the error is calculated as:

Error, E1 = Target (first pattern) – Output

Similarly for the second pattern, for experiment no.1, the test parameters are therefore:

Inputs $i_0 = 1$

$i_1 = 1$

$i_2 = 0$

$i_3 = 1$

Weight $w_0 = -2$

$w_1 = -2$

$w_2 = -2$

$w_3 = -2$

Output = 0

The sum and sigmoid output are calculated. Then the error is calculated as:

Error, E2 = Target (second pattern) – Output

The total error for experiment number 1 is denoted as TE1, and for experiment number 2 is TE2, and so on. Therefore, the total error (negative values were converted to positives) for experiment 1 would be:

TE1 = E1 + E2

The total error values for each experiment are given in Table 4-3.

Exp.No.	**Total Error**
Exp 1	0.9845
Exp 2	1.0000
Exp 3	1.3808
Exp 4	1.0000
Exp 5	0.2384
Exp 6	1.7616
Exp 7	0.5180
Exp 8	1.4820
Exp 9	1.0000

Shading indicates the lowest error

Table 4-3. Total error

Once all 9 experiments are completed, then "sum error" is calculated. Sum error is the total error for a particular weight, at a particular level. For example, the sum error for weight w_0 at level 1 can be calculated by adding up the total errors produced by the experiments 1, 2 and 3, which is given as:

Sum error at level 1 for weight w_0 = TE1 + TE2 + TE3

Similarly, Sum error at level 2 for weight w_0 = TE4 + TE5 + TE6

Sum error at level 3 for weight w_0 = TE7 + TE8 + TE9

Sum error may be calculated in a similar way for all the weights. Table 4-4 shows the sum error for all the weights.

	Level 1	Level 2	Level 3
Weight w_0	3.37	3.00	🟩 3.00
Weight w_1	🟩 2.50	2.72	4.14
Weight w_2	4.23	3.00	🟩 2.14
Weight w_3	🟩 2.22	3.28	3.86

Shading is used to indicate the lowest error for each weight

Table 4-4. Sum error

It can be seen that the lowest error for weight w_0 is level 3, for weight w_1 is level 1, for weight w_2 is level 3 and for weight w_3 is level 1. It is known that a lower error is better. Hence choose the best level (out of levels 1, 2 and 3) that produced the lowest sum error for weight w_0. The process is repeated for other weights. Therefore, the weights for the corresponding best levels are 2, -2, 2, -2, and that produced an error of **0.1372,** which is lower than the errors shown in Table 4-3.

4.2.2 Experiment Iterations

The best weights obtained in the first iteration often need to be refined, if the desired error was not reached at the first attempt. This can be done, for example, using Gradient Descent Algorithms (Haykin 1999) or methods like Simulated Annealing (Wasserman 1989). However, it is also possible to iterate the Taguchi Method and continue training the network in this way.

The best weights are selected from the first iteration and will be used as a starting point for subsequent iterations. To continue with the previous example, after the first iteration, the weights for the corresponding best levels were 2, -2, 2, -2 as shown in Figure 4-3.

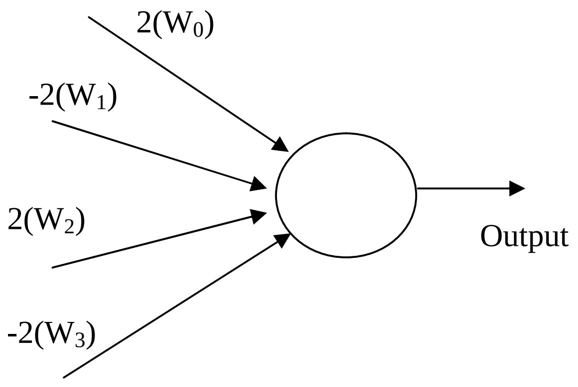


Figure 4-3

One can reapply the array and refine the weights further by reassigning the levels of the array to new weights. For example, in this case of weight w_0 (which is 2), we know that 2 is chosen in preference to other levels (0 and -2), so one can set the three levels for the next iteration to try 1, 2 and 3. Similarly, in the case of weight w_1 (which is -2), we know that -2 is chosen in preference to other levels (0 and 2), so one can try –3, -2 and -1 for these in the next attempt. The new levels for weights w_2 and w_3 are set in the same way. It can be seen that the range is also reduced to get closer to the optimum weights. Figure 4-4 shows the new levels and produces the array shown in Table 4-5.

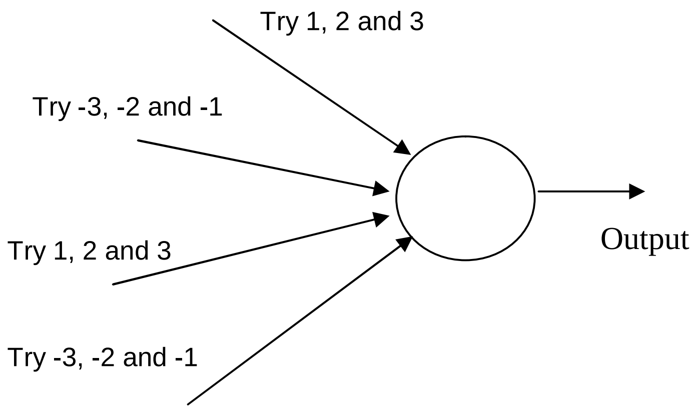


Figure 4-4. New levels

Exp.No.	Weight w_0	Weight w_1	Weight w_2	Weight w_3
Exp 1	1 (level 1)	-3 (level 1)	1 (level 1)	-3 (level 1)
Exp 2	1 (level 1)	-2 (level 2)	2 (level 2)	-2 (level 2)
Exp 3	1 (level 1)	-1 (level 3)	3 (level 3)	-1 (level 3)
Exp 4	2 (level 2)	-3 (level 1)	2 (level 2)	-1 (level 3)
Exp 5	2 (level 2)	-2 (level 2)	3 (level 3)	-3 (level 1)
Exp 6	2 (level 2)	-1 (level 3)	1 (level 1)	-2 (level 2)
Exp 7	3 (level 3)	-3 (level 1)	3 (level 3)	-2 (level 2)
Exp 8	3 (level 3)	-2 (level 2)	1 (level 1)	-1 (level 3)
Exp 9	3 (level 3)	-1 (level 3)	2 (level 2)	-3 (level 1)

Table 4-5. New levels for iteration 2

The experiments are conducted for iteration 2 in the same way as explained for iteration 1. It can be clearly seen that the error has reduced over the iteration for all the weights, which are shown in Figure 4-5.

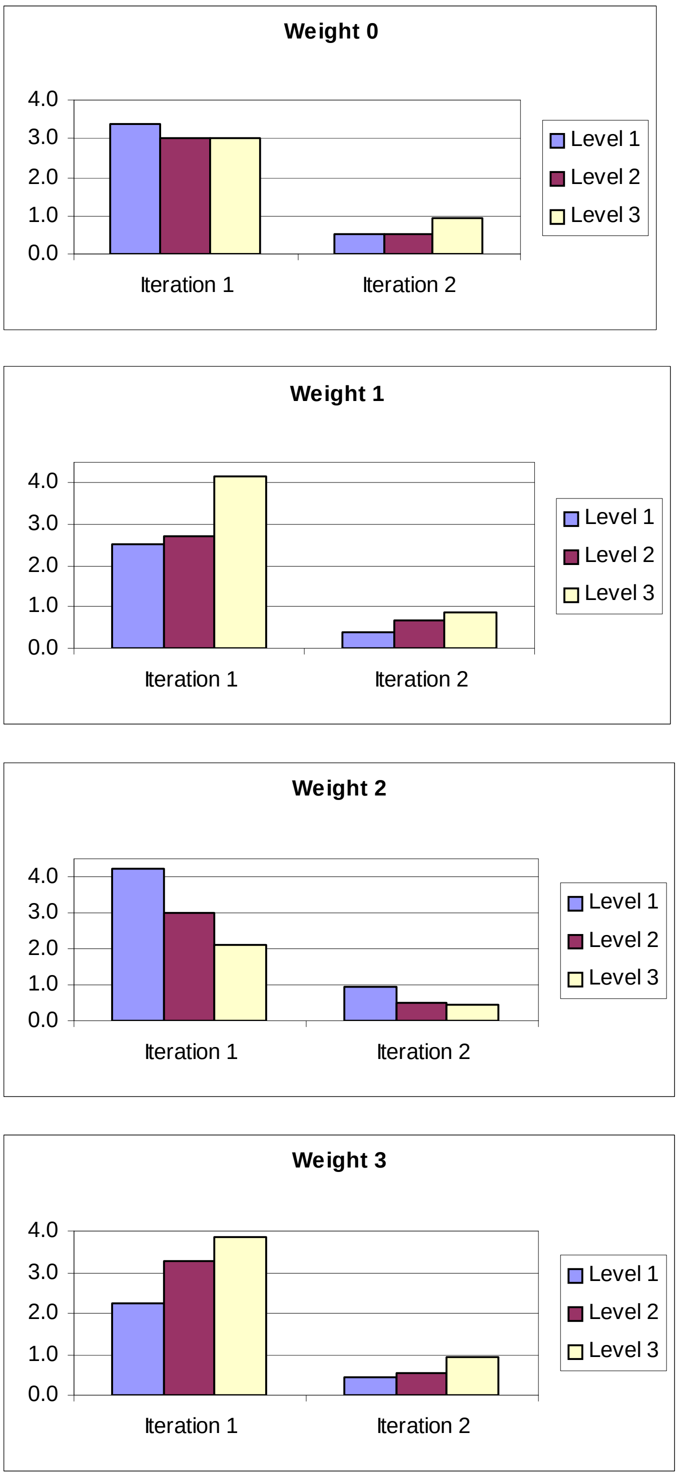


Figure 4-5. Sum error for weights with different iterations

After iteration 2, the weights for the corresponding best levels are 1, -3, 3, -3, and that produced an error of **0.0247,** which is *less than* the error obtained after iteration 1, using the best weights.

Figure 4-6 shows how the weight w_0 found its way to the optimum value in two iterations.

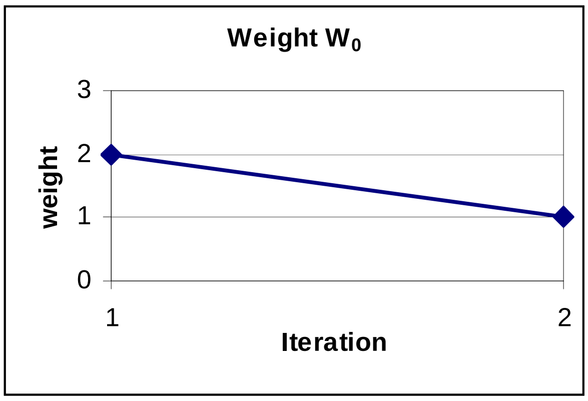


Figure 4-6. Trend of weight w_0 with different iterations

4.3 Training Single Layer Networks

In the previous section, it was shown that Taguchi methods may be applied to train simple neurons. In order to explore the capabilities of the Taguchi Methods of training, experiments were conducted on single layer neural networks for pattern recognition problems.

4.3.1 Experimental Design

Consider a simple single layer network with four neurons (A, B, C and D) as shown in Figure 4-7. Each neuron has four inputs, namely i_0, i_1, i_2 and i_3 and w_0, w_1, w_2 and w_3 are the corresponding weights for each neuron.

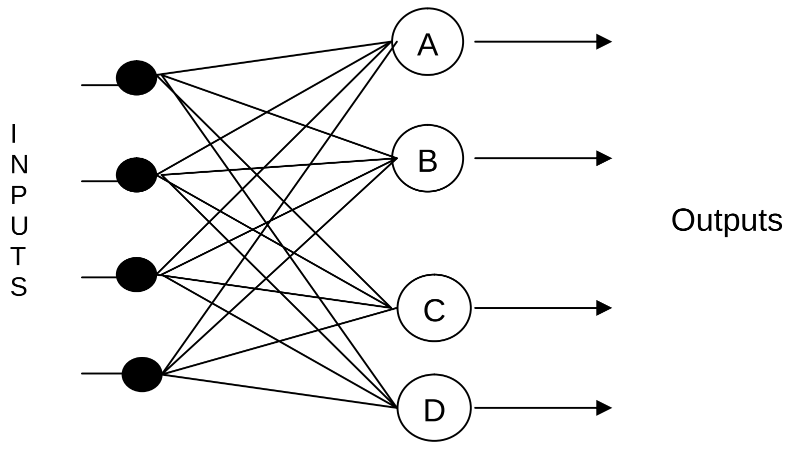


Figure 4-7. Single layer network

Each neuron has to be trained for a pattern. The following test parameters for four patterns are used to conduct the experiments, which are shown in Figure 4-8.

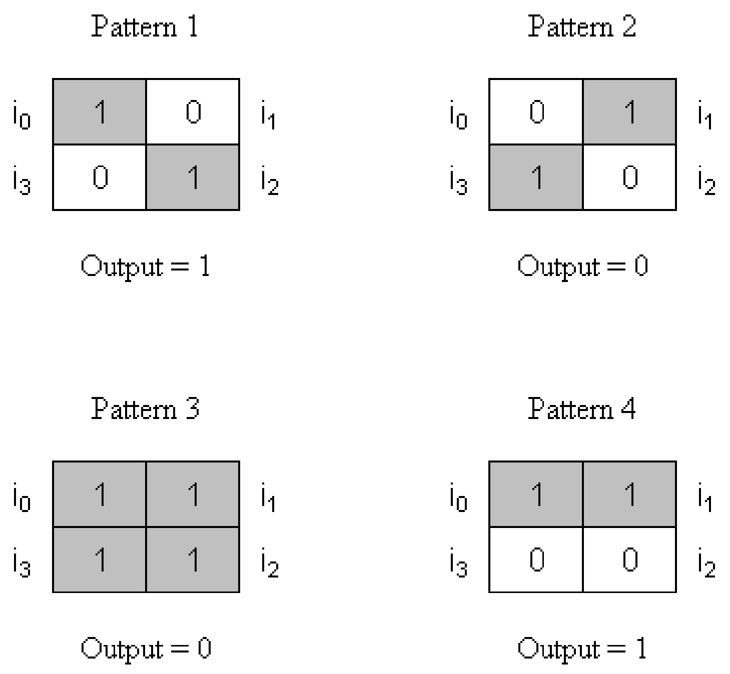


Figure 4-8. Patterns

An L9 orthogonal array may be used to conduct the experiments. Weights w_0, w_1, w_2 and w_3 are assigned to column 1, column 2, column 3 and column 4 of L9 OA respectively. The weights corresponding to the used levels are:

Level 1 = -2

Level 2 = 0

Level 3 = 2

Table 4-6 shows the array of experiments.

Experiment No.	Weight w_0	Weight w_1	Weight w_2	Weight w_3
Exp 1	-2	-2	-2	-2
Exp 2	-2	0	0	0
Exp 3	-2	2	2	2
Exp 4	0	-2	0	2
Exp 5	0	0	2	-2
Exp 6	0	2	-2	0
Exp 7	2	-2	2	0
Exp 8	2	0	-2	2
Exp 9	2	2	0	-2

Table 4-6 Experimental layout

As explained in previous section, all the experiments are conducted and the sum error is calculated. Table 4-7 shows the error for individual experiments for pattern 1, and the sum error is given in Table 4-8.

Exp. No.	Error
Exp 1	0.9820
Exp 2	0.8808
Exp 3	0.5000
Exp 4	0.5000
Exp 5	0.1192
Exp 6	0.8808
Exp 7	0.0180
Exp 8	0.5000
Exp 9	0.1192

Table 4-7 Experimental layout

	Level 1	Level 2	Level 3
Weight w_0	2.36	1.50	0.64
Weight w_1	1.50	1.50	1.50
Weight w_2	2.36	1.50	0.64
Weight w_3	1.22	1.78	1.50

Table 4-8 Sum error

Therefore, the weights for the corresponding best levels are 2, 0, 2, -2, and that produced an error of **0.0180,** which is exactly the same as the error produced by experiment number 7 (which produced the lowest error among other OA table of experiments).

In the same way all the patterns are trained and the errors are similar to those demonstrated for pattern 1. Figure 4-9 compares the lowest error obtained from the OA table of experiments with the error produced by the best weights. Best weights must produce the lowest error or at least the same error produced by the OA table of experiments. It can be seen that best weights produced the error which is the same as the error produced by the OA table of experiments. This is true for all the patterns trained. These results show that all patterns were trained successfully.

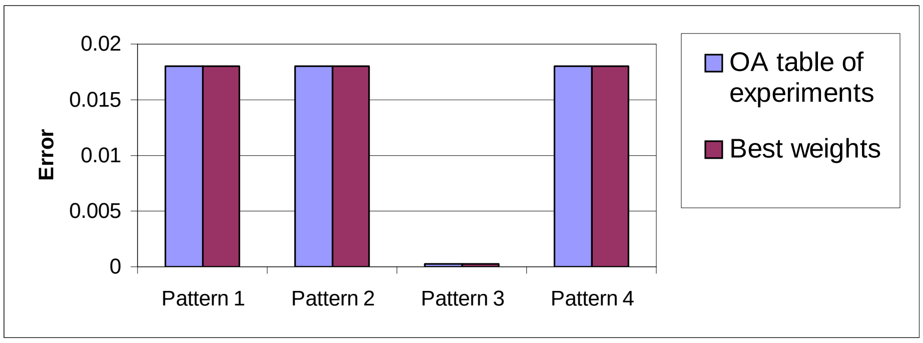


Figure 4-9. Comparison of errors

4.4 Training Multi-Layer Networks

In the previous sections of this chapter, it was shown that Taguchi methods may be applied to train simple neurons or single layer networks. In order to explore the capabilities and limitations of the Taguchi Methods of training further, experiments are conducted on multi-layer neural networks for pattern recognition problems.

4.4.1 Experimental Design

Figure 4-10 shows a simple multi-layer network used for experiments. It has two inputs i_1, i_2 and six weights w_1, w_2, w_3, w_4, w_5 and w_6. Taguchi orthogonal array L8 may be used to conduct the experiment. The L8 OA is a two level array (levels 1, and 2) with 8 experiments.

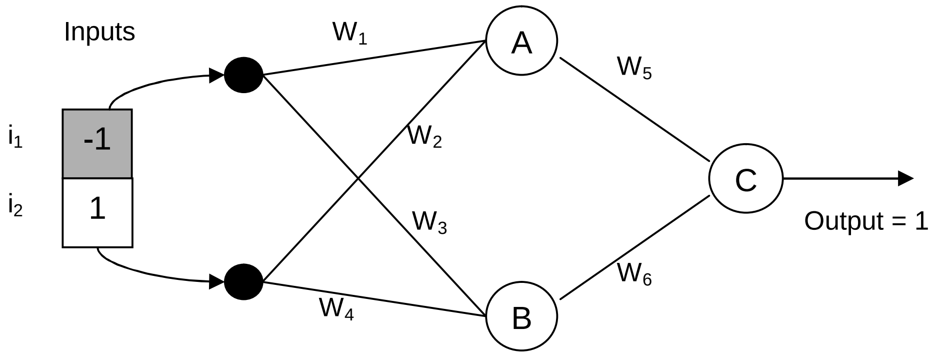


Figure 4-10. Multi-layer network

Weights w_1, w_2, w_3, w_4, w_5 and w_6 are assigned to column 1, column 2, column 3, column 4, column 5 and column 6 of the L8 OA respectively.

The weights corresponding to the levels are:

Level 1 = -2

Level 2 = 2

Therefore, the experimental layout, which is used to conduct experiments with actual weights, is shown in Table 4-9.

Exp.No.	Weight w_0	Weight w_1	Weight w_2	Weight w_3	Weight w_4	Weight w_5
Exp 1	-2	-2	-2	-2	-2	-2
Exp 2	-2	-2	-2	2	2	2
Exp 3	-2	2	2	-2	-2	2
Exp 4	-2	2	2	2	2	-2
Exp 5	2	-2	2	-2	2	-2
Exp 6	2	-2	2	2	-2	2
Exp 7	2	2	-2	-2	2	2
Exp 8	2	2	-2	2	-2	-2

Table 4-9. L8 Orthogonal Array experimental layout

As explained in section 4.2.1, all 8 experiments are conducted and the error is calculated which is shown in Table 4-10. Then the sum error is calculated, which is shown in Table 4-11.

Exp. No.	**Error**
Exp 1	0.8808
Exp 2	0.0491
Exp 3	0.8730
Exp 4	0.2761
Exp 5	0.5000
Exp 6	0.2761
Exp 7	0.1192
Exp 8	0.9509

Shading is used to indicate the lowest error

Table 4-10. Individual error for experiments

	Weight w_0	Weight w_1	Weight w_2	Weight w_3	Weight w_4	Weight w_5
Level 1	2.0790	1.7059	2.0000	2.3730	2.9808	2.6078
Level 2	1.8462	2.2192	1.9252	1.5521	0.9444	1.3174

Shading is used to indicate the lowest error for each weight

Table 4-11. Sum error

As explained earlier, we must choose the best level (out of levels 1 and 2) that produces the lowest sum error for the weights.

Therefore, the weights for the corresponding best levels are 2, -2, 2, 2, 2, 2, which produces an error of **0.2619**. It can be noticed that the error is *increased* when compared to the lowest error shown in Table 4-10, and is represented in graphical form in Figure 4-11.

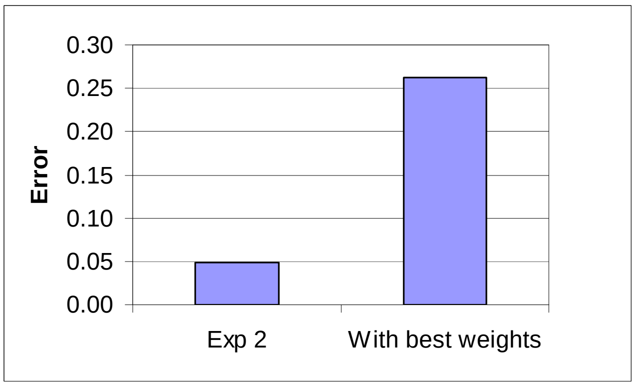


Figure 4-11. Comparison of errors

This is due to training weights in different layers at the same time. When the network is trained as a whole, the weights in different layers interact; for example, the weights in the first layer interact strongly with those in second layer (that is, if one changes the first layer weights, those in the second layer must also change, if the error is to stay the same or to reduce further). The basic Taguchi Method can only handle a small amount of interaction between factors as it primarily focuses on the main effects of the factors. This, therefore, causes the training to fail.

4.5 Summary

It was shown that Taguchi Methods may be applied to train a single neuron or a single layer network. However, when the method was applied to the multi-layer networks for pattern recognition problems, the network performs poorly due to the interaction between the weights in different layers.

To overcome this, alternate ideas have been developed to train the multi-layer networks using Taguchi Methods, which are discussed in the next chapter.

Chapter 5

New Training Methods using Taguchi Methods for ANN Training

5.1 Introduction to Chapter

In the previous chapter, it was shown that Taguchi Methods may be used to train single layer networks. It was also highlighted that this method of training does not work well in multi-layer networks due to interaction between the layers. Due to the potential capabilities of Taguchi Methods for ANN training, it was decided to explore this further.

Based on this, a number of ideas were developed including:

- Layer-By-Layer training method
- Neuron-By-Neuron training method
- Using Polynomials

This chapter discusses the new techniques developed to train multi-layer neural networks using Taguchi Methods. Each of the above methods is discussed in the following subsections.

5.2 Layer-By-Layer Training Method

To overcome the problem of interaction between the weights in different layers, the following method of training was tried. Consider the following network with two layers (1 and 2) as shown in Figure 5.1.

The idea is to apply Taguchi Methods initially to train the weights in layer 1 only (layer 2 weights are set to a known value of ‘1’). Once the weights in layer 1 have been fixed, Taguchi Methods is then applied to layer 2. This way, both layers of weights can be trained by Taguchi Methods, one at a time.

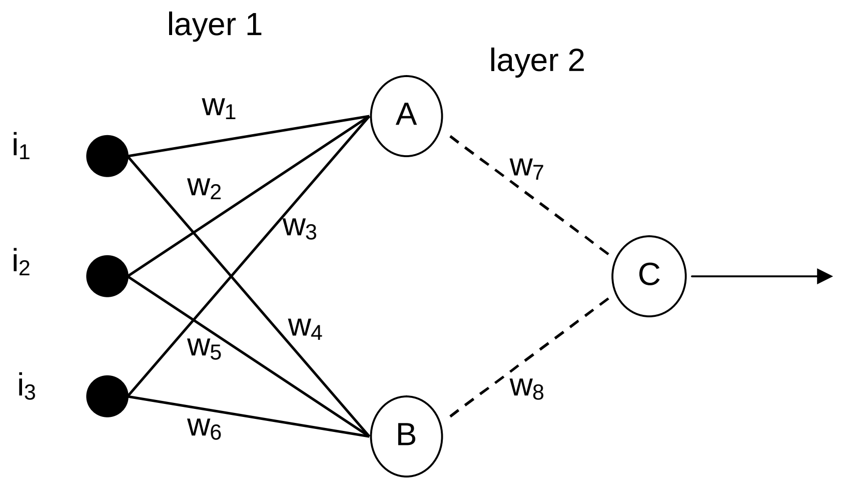
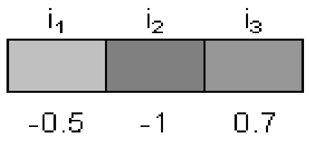


Figure 5-1. Training multi layer networks

The method illustrated above was employed to train the network shown in Figure 5-1. The weights for layer 1 (from inputs to neurons A and B) are denoted as w_1, w_2, w_3, w_4. w_5 and w_6 and the weights for layer 2 (from neuron A and B to neuron C) are w_7 and w_8.

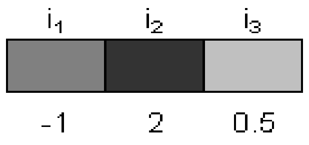
The following patterns were used to train the network:

Pattern 1



Output = 1

Pattern 2



Output = 1

To train this network an L8 OA (two levels and 8 trials) may be used; this is shown in Table 5-1.

Expt. No.	Col.1	Col.2	Col.3	Col.4	Col.5	Col.6	Col.7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

Table 5-1. L8 Orthogonal Array

Weights w_1, w_2, w_3, w_4. w_5 and w_6 are assigned to column 1, column 2, column 3, column 4, column 5 and column 6 of L8 OA respectively. The weights corresponding to the used levels were:

Level 1 = -1

Level 2 = 1

As the weights in layer 1 are being trained, weights in layer 2 (w_7 and w_8) are set to a known value of '1' to eliminate the effect of these weights on the other layer. Therefore, the experimental layout for weights in layer 1, which is used to conduct experiments with actual weights, is shown in Table 5-2.

	W1	W2	W3	W4	W5	W6
Exp 1	-1	-1	-1	-1	-1	-1
Exp 2	-1	-1	-1	1	1	1
Exp 3	-1	1	1	-1	-1	1
Exp 4	-1	1	1	1	1	-1
Exp 5	1	-1	1	-1	1	-1
Exp 6	1	-1	1	1	-1	1
Exp 7	1	1	-1	-1	1	1
Exp 8	1	1	-1	1	-1	-1

Table 5-2. L8 Orthogonal Array experimental layout

Network error calculations were done as explained before in the previous chapter (section 4.2.1). Table 5-3 shows the sum error matrix. Shade indicates the lowest error.

	W1	W2	W3	W4	W5	W6
Level 1	2.0569	2.3257	2.3692	2.0569	2.3545	2.3692
Level 2	2.3975	2.1287	2.0852	2.3975	2.0998	2.0852

Table 5-3. Sum error for layer 1 weights

Based on the lowest errors, the best levels are selected for layer 1 weights, which are, for this example:

-1, 1, 1, -1, 1, 1

The layer 1 weights are fixed, and the weights for layer 2 need to be trained and fixed now. In order to do this, the levels for weights w_7 and w_8 are to be chosen. If one has to consider the levels as 0 and 1, then all possible four combinations would be:

0 0, 0 1, 1 0, 1 1

which is shown as:

	W7	W8
Exp 1	0	0
Exp 2	0	1
Exp 3	1	0
Exp 4	1	1

Then the experiment layout would look as shown in Table 5-4. It may be noted that layer 1 weights w_1, w_2, w_3, w_4. w_5 and w_6 are already selected.

	W1	W2	W3	W4	W5	W6	W7	W8
Exp 1	-1	1	1	-1	1	1	0	0
Exp 2	-1	1	1	-1	1	1	0	1
Exp 3	-1	1	1	-1	1	1	1	0
Exp 4	-1	1	1	-1	1	1	1	1

Table 5-4. Experiment layout to fix layer 2 weights

Table 5-5 shows the sum error matrix to observe the lowest errors for layer 2 weights w_7 and w_8. Shading indicates the lowest error. Based on the lowest error, the weights for the corresponding best levels are selected.

	Error
Exp 1	1.0000
Exp 2	0.6406
Exp 3	0.6406
Exp 4	0.3753

Table 5-5. Sum error

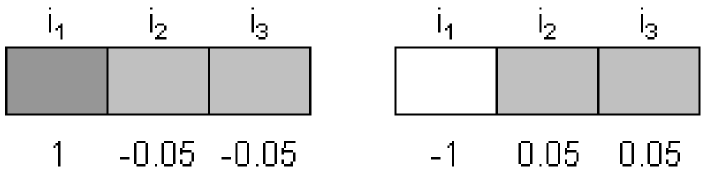
Therefore the best weights are -1, 1, 1, -1, 1, 1, 1, 1 (which produced an error of 0.3753).

If one conducts the full factorial experiments to find out the lowest error possible, 256 tests need to be performed (8 factors each at two levels). To compare the lowest error achieved through the layer-by-layer training method, all the 256 tests are performed and the errors are calculated. The lowest error obtained is 0.3753 which is *exactly the same*, when compared with the layer-by-layer training method.

Limitations

Although layer-by-layer method of training successfully recognised the patterns for the above example, when it was tested on some other patterns (an example is given below), it was unable to produce the lowest error possible.

Pattern 1 Pattern 2



Output = 1 Output = 1

For these patterns, layer-by-layer training method produced an error of 0.5552, but the full factorial experiment produced a lowest error of 0.538. There are few experiments which produced this lowest error, for example, experiment no. 242 (Full results are given in Appendix A3).

When all the weights in layer 1 are being trained, the weights which are connected to different neurons interact; for example, the weights which are connected to neuron A interact with those which are connected to neuron B (that is, if one changes the neuron A weights, those in the neuron B must also change, if the error is to stay the same or to reduce further). The basic Taguchi Method can only handle a small amount of interaction between factors as it primarily focuses on the main effects of the factors. This, therefore, causes the training to fail. This also depends on the complexity of the patterns being trained.

The other aspect is, although it is possible to use this method for large networks, selecting the suitable OA becomes difficult.

5.3 Neuron-By-Neuron Training Method

As part of the effort to develop a training method which will use Taguchi Methods to train multi-layer ANNs, another idea was developed. Consider the network given in Figure 5-2.

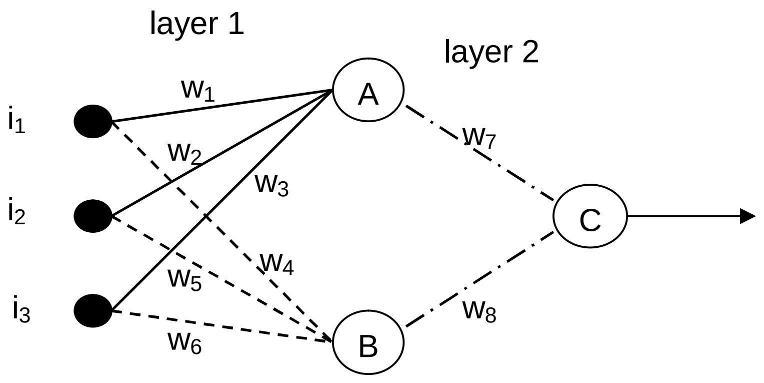


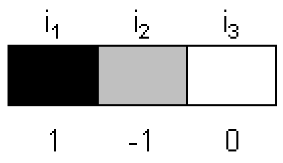
Figure 5.2. Training multi layer networks

The idea is, first train and fix the weights w_1, w_2 and w_3 for neuron A (weights w_4, w_5 and w_6 for neuron B are set to '0', and the layer 2 weights w_7 and w_8 are set to a known value of '1'). Then in the same way, fix the weights w_4, w_5 and w_6 for neuron B, and finally fix the weights w_7 and w_8 for layer 2. This method of training a set of weights which are connected to a particular neuron for a particular layer, avoids the interlayer-weights interaction.

The method illustrated above was employed to train the network shown in Figure 5-2. To train this network L4 OA (two levels and 4 runs) may be used which is shown in Table 5-6.

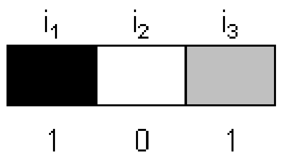
The following patterns were used to train the network:

Pattern 1:



Output = 1

Pattern 2:



Output = 1

Experiment Number	Column 1	Column 2	Column 3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Table 5-6. L4 Orthogonal Array

Weights w_1, w_2 and w_3 are assigned to column 1, column 2 and column 3 of L4 OA respectively.

The weights corresponding to the used levels are:

Level 1 = -1

Level 2 = 1

When the weights (w_1, w_2 and w_3) for neuron A are trained, weights (w_4. w_5 and w_6) for neuron B are set to '0', and the layer 2 weights (w_7 and w_8) are set to a known value of '1' to eliminate the effect of these weights on the other layer.

Therefore, the experimental layout for weights w_1, w_2 and w_3 for neuron A is shown in Table 5-7.

W1	W2	W3
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Table 5-7. L4 Orthogonal Array experimental layout

Table 5-8 shows the sum error matrix. Shading indicates the lowest error.

	W1	W2	W3
Level 1	1.2378	1.0207	1.1568
Level 2	0.9396	1.1568	1.0207

Table 5-8. Sum error for neuron A weights

Based on the lowest errors, the best levels are selected for neuron A weights, which are, for this example:

1, -1, 1

In the same way, using the weights for the corresponding best levels for neuron A, weights for neuron B is fixed (keeping the layer 2 weights as '1'). Table 5-9 shows the sum error matrix. Shading indicates the lowest error.

	W4	W5	W6
Level 1	0.9396	0.7630	0.8716
Level 2	0.6949	0.8716	0.7630

Table 5-9. Sum error for neuron B weights

Based on the lowest errors, the best levels are selected for neuron B weights, which are, in this case:

1, -1, 1

Now, the weights for neuron A and neuron B are fixed, we need to find the best levels for weights w_7 and w_8 in layer 2. As explained before, if we consider the levels as 0 and 1, then all the possible combinations are:

0 0, 0 1, 1 0, 1 1

which may be shown as:

	W7	W8
Exp 1	0	0
Exp 2	0	1
Exp 3	1	0
Exp 4	1	1

The experimental layout would then look as shown in Table 5-10. It may be noted that layer 1 weights w_1, w_2, w_3, w_4. w_5 and w_6 are the best levels, already selected.

	W1	W2	W3	W4	W5	W6	W7	W8
Exp 1	1	-1	-1	-1	1	1	0	0
Exp 2	1	-1	-1	-1	1	1	0	1
Exp 3	1	-1	-1	-1	1	1	1	0
Exp 4	1	-1	-1	-1	1	1	1	1

Table 5-10. Experiment layout to fix layer 2 weights

Table 5-11 shows the sum error matrix used to observe the lowest errors for layer 2 weights w_7 and w_8. Shading indicates the lowest error. Based on the lowest error, the weights for the corresponding best levels are selected.

	Error
Exp 1	1.0000
Exp 2	0.5860
Exp 3	0.5860
Exp 4	0.293

Table 5-11. Sum error

Therefore, the best weights are 1, -1, 1, 1, -1, 1, 1, 1 (which produced an error of 0.293).

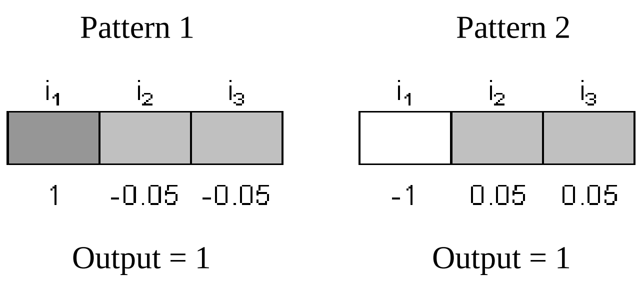
Comparing the full factorial experiment results shown in Appendix A4, experiment number 238 produced the lowest error of 0.293 which is *exactly the same* as that obtained using the Neuron-By-Neuron training method.

Alternatively, this method can also be used for training in a slightly different way. By including the output weight of the neuron being trained in the array (OA table), it is possible to train, at the same time, all the weights which are connected to a particular neuron. This method also successfully recognises the patterns and finds the lowest error possible.

Advantages

The Neuron-By-Neuron training method has been tested with several patterns and it was able to successfully recognise all of them. It is a very consistent and reliable method of training.

For example, the same pattern, which is given below (which layer-by-layer training method failed to produce the lowest error) was tested with this method. It successfully produced the lowest error and is comparable with the full factorial results given in Appendix A3.



Limitations

This method of training is more suitable for relatively small networks. However, it is possible to use this method for large networks by selecting or generating the suitable OAs.

5.4 Using Polynomials

The research team at The Robert Gordon University has produced a new neural model based on the idea that a neural unit should be flexible enough to fulfil any mathematical function required of it (McMinn 2002).

The most common artificial neural models in current use are those developed from the original McCulloch-Pitts model (Wasserman 1989). Ignoring the squashing or activation function, which normalises the output, the activity of this neuron is given by:

$$A = \sum_{i=1}^{n} x_i w_i$$

Where n is the number of inputs, x_i is an input and w_i is its corresponding weight.

For a two input neuron, with input x associated with weight b and input y associated with c, as illustrated in Figure 5-3, the activity could be written as:

$$A = bx + cy$$

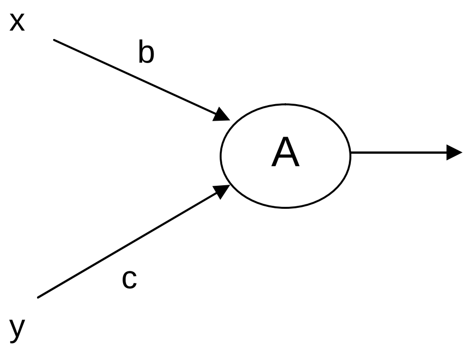


Figure 5-3. A simple neuron

This corresponds to a linear separator (Khanna 1990).

One can model any continuous function using an infinite Power Series (for example, a Taylor series):

$$f(x) = bx + cx^2 + \text{.......... ....} + \delta_n x^{n-1}$$

This is the basic series, which is given in most references. However, it is extendable to any number of variables (and hence any number of dimensions). For example, in two dimensions (or, for a two input neuron), the series is:

$$A = (b_1 x + c_1 y) + (b_2 x^2 + c_2 y^2) + (b_3 x^3 + c_3 y^3) + \text{.............} + b_n x^n + c_n y^n$$

Taguchi Methods may be used to train such networks because one can set the first order weights initially, then second order weights (squared terms), and then the third order weights (cubed terms) etc. The network error reduces with each increasing input power (as the mathematical approximation becomes more accurate).

The method illustrated above was employed to train the neuron shown in Figure 5-4. To train this neuron, an L4 OA (two levels and 4 trials) may be used.

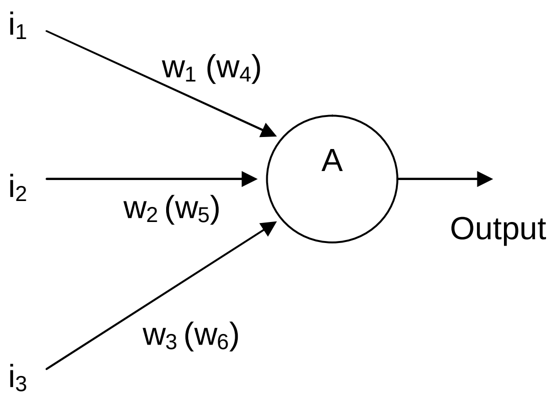
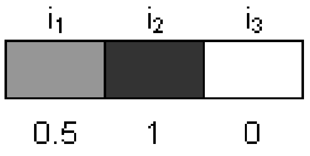
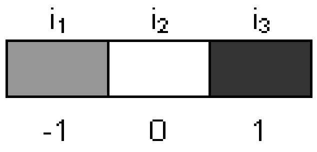


Figure 5-4. A simple neuron with three inputs

The following test parameters are used to conduct the experiments:



Output = 1



Output = 1

Weights w_1, w_2 and w_3 are assigned to column 1, column 2 and column 3 of L4 OA respectively.

The weights corresponding to the levels used are:

Level 1 = -1

Level 2 = 1

Therefore, the experimental layout, which is used to conduct experiments, is shown in Table 5-12.

	W1	W2	W3
Exp 1	-1	-1	-1
Exp 2	-1	1	1
Exp 3	1	-1	1
Exp 4	1	1	-1

Table 5-12. L4 Orthogonal Array experimental layout

All the experiments are conducted and the first order output is calculated. Based on the lowest error, the weights for the corresponding best levels are chosen, and then the optimum output is calculated.

For the second order (squared terms), the inputs are squared and then the output is calculated using the sum previously obtained from the first order trial (the second order weights are shown in brackets in figure 5-4). For example, the second order output may be written as,

$$Sum = i_1 w_1 + i_2 w_2 + i_3 w_3 + i_1^2 w_4 + i_2^2 w_5 + i_3^2 w_6$$

As explained previously, based on the lowest error, the weights for the corresponding best levels are chosen, and then the optimum output is calculated for the second order. The same approach can be used for the third order (cubed terms).

Figure 5-5 shows how the error reduces when the order increases as the approximation becomes more accurate.

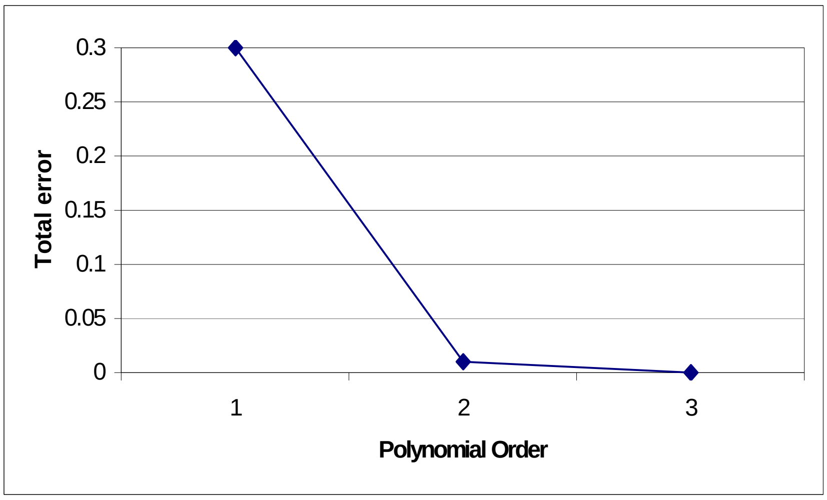


Figure 5-5. Error for the network

Using factorials in the series may improve the accuracy of the solution by scaling the higher order terms. The factorial may be multiplied or divided depending on the weights.

For example, if 0<w<1 (or) 0>w>-1 then multiply by the factorial for that order. For the second order, the sum may be written as,

$$Sum = (i_1 w_1 + i_2 w_2 + i_3 w_3) + (i_1^2 w_4 \times 2! + i_2^2 w_5 \times 2! + i_3^2 w_6 \times 2!)$$

and if w>1 (or) w<-1 then divide by the factorial for that order. This may be applied in a similar way to third order and so on. Figure 5-6 shows reduction on error with factorials.

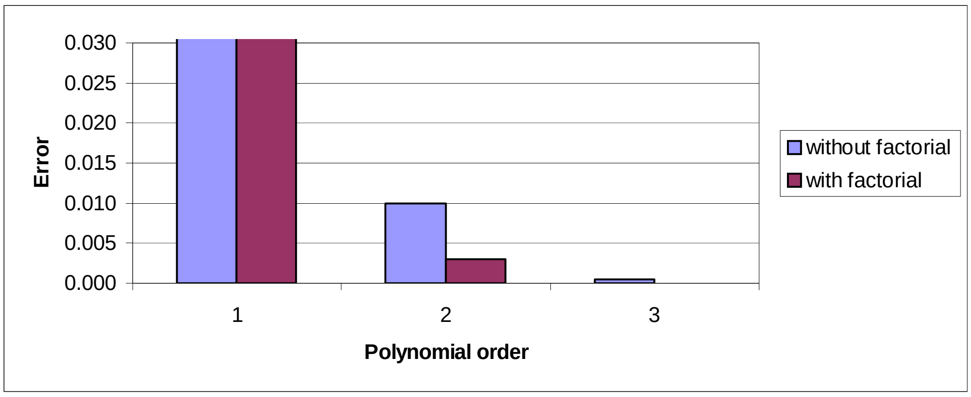


Figure 5-6. Using factorials in power series

It is also possible to use Polynomials with the Neuron-By-Neuron method of training demonstrated in section 5.3, so that each polynomial neuron can be trained, one at a time. For example, consider the network shown in Figure 5-7.

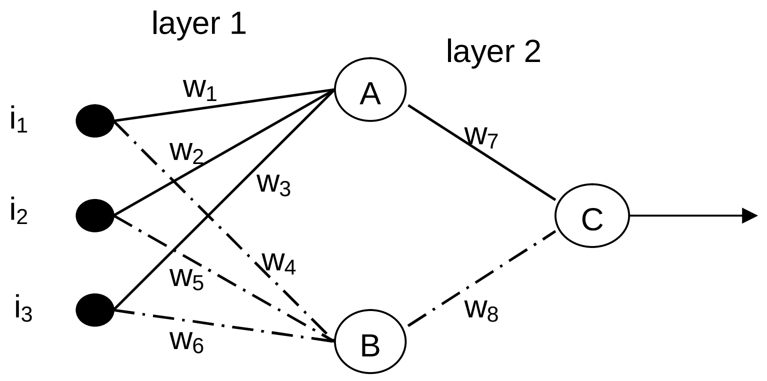


Figure 5-7. Multi-layer network

Weights w1, w2, w3 and w7, which are connected to the neuron A, can be trained at the same time but training as a polynomial neuron. It is possible to iterate this method to reduce the error to the desired level.

Advantages

Training using Polynomial neurons successfully recognises and differentiates the patterns being trained. A neuron can function in a single layer and the approximation becomes more accurate with the increasing order.

Limitations

This method of training is suitable for a network size which is relatively small. Although it is possible to use this method for large networks, selecting the suitable OA may be difficult under some circumstances.

Chapter 6

New Training Method using Custom-Made OAs for ANN Training

6.1 Introduction to Chapter

In the previous chapters, it has been demonstrated that Taguchi Methods may be applied to train a single neuron, single layer networks and multi-layer networks successfully. These networks are usually small, with a fixed network structure. It has been highlighted that selecting a suitable OA table for a given problem may be difficult under some circumstances.

Therefore, another new method of training was also developed that uses custom-built OA tables. This chapter discusses a new method called ‘Coding the state of each neuron’ training.

6.2 Coding the State of Each Neuron

This method of training is based on the idea that each level in the OA table may correspond to the state of an individual neuron. Consider the multi layer network shown in Figure 6-1.

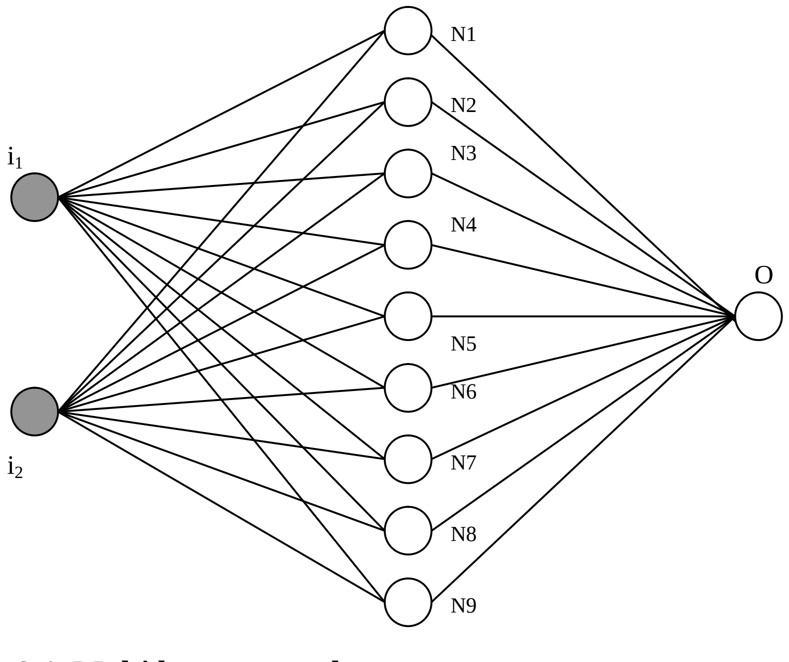


Figure 6-1. Multi layer network

As can be seen from the network structure, considering the number of weights there is no standard published OA available to fit it. As explained previously in Chapter 3, published Orthogonal Arrays are of a fixed and often inconvenient size. Also, very large OAs are not often published and these may be needed for larger networks.

Hence, we need to use a custom-built table to train this multi-layer network. The method used to generate the matrices, for this particular problem, was based on algorithms by (Owen 2004).

The coding of a neuron is explained by referring to Figure 6-2, levels are shown in Table 6-1, and experiments are shown in Table 6-2. Each neuron may take 8 different levels, from 0 to 7 (based on the experiments). For example, for neuron N1, if the level is ‘0’ for the first experiment (refer Table 6-2), then the weights corresponding to the level are –1, -1, -1 which is illustrated in Figure 6-2. Similarly for neuron N2 (which is level 5 for the first experiment), then the weights are 1, -1, 1 and so on, which are highlighted in Table 6-1.

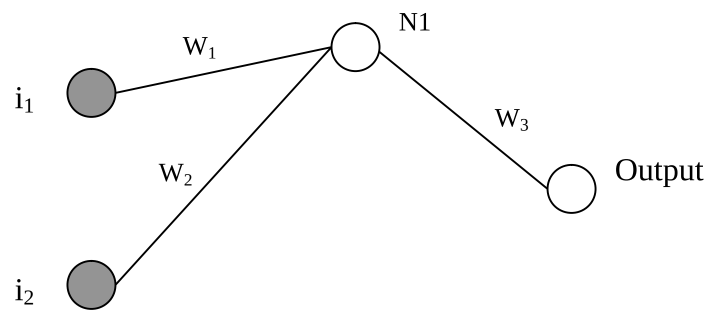


Figure 6-2. Neuron N1 with level 0

Weight	LEVELS							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
W_1	-1	1	-1	1	-1	1	-1	1
W_2	-1	-1	1	1	-1	-1	1	1
W_3	-1	-1	-1	-1	1	1	1	1

Table 6-1. Levels for experiments

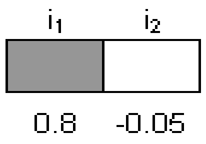
9 neurons are assigned to 9 columns of the OA table as shown in Table 6-2. Total number of experiments is 64.

Table 6-2. OA table for experiments

	NEURONS								
	N1	N2	N3	N4	N5	N6	N7	N8	N9
Exp 1	0	5	6	0	1	2	2	5	1
Exp 2	0	6	4	4	2	7	3	0	7
Exp 3	0	4	2	3	3	4	6	3	2
Exp 4	0	7	3	6	4	0	1	7	3
Exp 5	0	3	7	1	0	5	0	6	4
Exp 6	0	0	1	5	6	1	7	4	0
Exp 7	0	2	0	7	7	6	4	2	5
Exp 8	0	1	5	2	5	3	5	1	6
Exp 9	7	5	4	3	4	5	7	2	6
Exp 10	7	6	6	6	3	1	0	1	5
Exp 11	7	4	3	0	2	6	5	6	0
Exp 12	7	7	2	4	1	3	4	4	4
Exp 13	7	3	1	7	5	2	3	3	3
Exp 14	7	0	7	2	7	7	2	7	2
Exp 15	7	2	5	1	6	4	1	5	7
Exp 16	7	1	0	5	0	0	6	0	1
Exp 17	5	5	2	1	7	1	5	0	3
Exp 18	5	6	3	5	5	5	4	5	2
Exp 19	5	4	6	7	0	3	7	7	7
Exp 20	5	7	4	2	6	6	0	3	1
Exp 21	5	3	0	0	3	7	1	4	6
Exp 22	5	0	5	4	4	2	6	6	5
Exp 23	5	2	7	3	1	0	3	1	0
Exp 24	5	1	1	6	2	4	2	2	4
Exp 25	2	5	3	7	6	7	6	1	4
Exp 26	2	6	2	2	0	2	1	2	0
Exp 27	2	4	4	1	5	0	2	4	5
Exp 28	2	7	6	5	7	4	3	6	6
Exp 29	2	3	5	3	2	1	4	7	1
Exp 30	2	0	0	6	1	5	5	3	7
Exp 31	2	2	1	0	4	3	0	0	2
Exp 32	2	1	7	4	3	6	7	5	3
Exp 33	3	5	7	5	2	3	1	3	5
Exp 34	3	6	1	1	1	6	6	7	6
Exp 35	3	4	0	2	4	1	3	5	4
Exp 36	3	7	5	7	3	5	2	0	0
Exp 37	3	3	6	4	6	0	5	2	2
Exp 38	3	0	4	0	0	4	4	1	3
Exp 39	3	2	2	6	5	7	7	6	1
Exp 40	3	1	3	3	7	2	0	4	7
Exp 41	4	5	1	2	3	0	4	6	7
Exp 42	4	6	7	7	4	4	5	4	1
Exp 43	4	4	5	5	1	7	0	2	3
Exp 44	4	7	0	1	2	2	7	1	2

Exp 45	4	3	4	6	7	3	6	5	0
Exp 46	4	0	6	3	5	6	1	0	4
Exp 47	4	2	3	4	0	1	2	3	6
Exp 48	4	1	2	0	6	5	3	7	5
Exp 49	6	5	0	4	5	4	0	7	0
Exp 50	6	6	5	0	7	0	7	3	4
Exp 51	6	4	7	6	6	2	4	0	6
Exp 52	6	7	1	3	0	7	5	5	5
Exp 53	6	3	2	5	4	6	2	1	7
Exp 54	6	0	3	1	3	3	3	2	1
Exp 55	6	2	6	2	2	5	6	4	3
Exp 56	6	1	4	7	1	1	1	6	2
Exp 57	1	5	5	6	0	6	3	4	2
Exp 58	1	6	0	3	6	3	2	6	3
Exp 59	1	4	1	4	7	5	1	1	1
Exp 60	1	7	7	0	5	1	6	2	7
Exp 61	1	3	3	2	1	4	7	0	5
Exp 62	1	0	2	7	2	0	0	5	6
Exp 63	1	2	4	5	3	2	5	7	4
Exp 64	1	1	6	1	4	7	4	3	0

Once the levels and weights are chosen for the neurons, then the network was trained. The following test parameters were used:



Output =1

Once the final output is calculated for experiment 1, then the error is calculated using the target output. Similarly, the error is calculated for all the experiments which are shown in Table 6-3. It can be observed that experiment number 18 which produced a lowest error of 0.04. Then sum error was calculated for each level which is shown in Table 6-4. These calculations are explained previously in Chapter 4.

Table 6-3. Experiments showing errors

	NEURONS									Error
	N1	N2	N3	N4	N5	N6	N7	N8	N9	
Exp 1	0	5	6	0	1	2	2	5	1	**0.72**
Exp 2	0	6	4	4	2	7	3	0	7	**0.34**
Exp 3	0	4	2	3	3	4	6	3	2	**0.88**
Exp 4	0	7	3	6	4	0	1	7	3	**0.67**
Exp 5	0	3	7	1	0	5	0	6	4	**0.58**
Exp 6	0	0	1	5	6	1	7	4	0	**0.59**
Exp 7	0	2	0	7	7	6	4	2	5	**0.19**
Exp 8	0	1	5	2	5	3	5	1	6	**0.57**
Exp 9	7	5	4	3	4	5	7	2	6	**0.06**
Exp 10	7	6	6	6	3	1	0	1	5	**0.53**
Exp 11	7	4	3	0	2	6	5	6	0	**0.34**
Exp 12	7	7	2	4	1	3	4	4	4	**0.28**
Exp 13	7	3	1	7	5	2	3	3	3	**0.84**
Exp 14	7	0	7	2	7	7	2	7	2	**0.10**
Exp 15	7	2	5	1	6	4	1	5	7	**0.16**
Exp 16	7	1	0	5	0	0	6	0	1	**0.73**
Exp 17	5	5	2	1	7	1	5	0	3	**0.48**
Exp 18	5	6	3	5	5	5	4	5	2	**0.04**
Exp 19	5	4	6	7	0	3	7	7	7	**0.05**
Exp 20	5	7	4	2	6	6	0	3	1	**0.43**
Exp 21	5	3	0	0	3	7	1	4	6	**0.67**
Exp 22	5	0	5	4	4	2	6	6	5	**0.06**
Exp 23	5	2	7	3	1	0	3	1	0	**0.91**
Exp 24	5	1	1	6	2	4	2	2	4	**0.66**
Exp 25	2	5	3	7	6	7	6	1	4	**0.21**
Exp 26	2	6	2	2	0	2	1	2	0	**0.93**
Exp 27	2	4	4	1	5	0	2	4	5	**0.32**
Exp 28	2	7	6	5	7	4	3	6	6	**0.09**
Exp 29	2	3	5	3	2	1	4	7	1	**0.84**
Exp 30	2	0	0	6	1	5	5	3	7	**0.49**
Exp 31	2	2	1	0	4	3	0	0	2	**0.95**
Exp 32	2	1	7	4	3	6	7	5	3	**0.42**
Exp 33	3	5	7	5	2	3	1	3	5	**0.56**
Exp 34	3	6	1	1	1	6	6	7	6	**0.71**
Exp 35	3	4	0	2	4	1	3	5	4	**0.73**
Exp 36	3	7	5	7	3	5	2	0	0	**0.39**
Exp 37	3	3	6	4	6	0	5	2	2	**0.66**
Exp 38	3	0	4	0	0	4	4	1	3	**0.89**
Exp 39	3	2	2	6	5	7	7	6	1	**0.34**
Exp 40	3	1	3	3	7	2	0	4	7	**0.84**
Exp 41	4	5	1	2	3	0	4	6	7	**0.42**
Exp 42	4	6	7	7	4	4	5	4	1	**0.05**
Exp 43	4	4	5	5	1	7	0	2	3	**0.33**
Exp 44	4	7	0	1	2	2	7	1	2	**0.72**
Exp 45	4	3	4	6	7	3	6	5	0	**0.28**
Exp 46	4	0	6	3	5	6	1	0	4	**0.52**
Exp 47	4	2	3	4	0	1	2	3	6	**0.88**
Exp 48	4	1	2	0	6	5	3	7	5	**0.33**
Exp 49	6	5	0	4	5	4	0	7	0	**0.11**

Exp 50	6	6	5	0	7	0	7	3	4	**0.16**
Exp 51	6	4	7	6	6	2	4	0	6	**0.13**
Exp 52	6	7	1	3	0	7	5	5	5	**0.11**
Exp 53	6	3	2	5	4	6	2	1	7	**0.42**
Exp 54	6	0	3	1	3	3	3	2	1	**0.99**
Exp 55	6	2	6	2	2	5	6	4	3	**0.42**
Exp 56	6	1	4	7	1	1	1	6	2	**0.82**
Exp 57	1	5	5	6	0	6	3	4	2	**0.42**
Exp 58	1	6	0	3	6	3	2	6	3	**0.92**
Exp 59	1	4	1	4	7	5	1	1	1	**0.81**
Exp 60	1	7	7	0	5	1	6	2	7	**0.27**
Exp 61	1	3	3	2	1	4	7	0	5	**0.84**
Exp 62	1	0	2	7	2	0	0	5	6	**0.64**
Exp 63	1	2	4	5	3	2	5	7	4	**0.32**
Exp 64	1	1	6	1	4	7	4	3	0	**0.81**

Level	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**	**N9**
0	4.552	4.276	4.565	4.317	4.595	4.519	4.415	4.377	4.359
1	5.044	5.193	5.098	4.778	5.091	5.143	5.142	5.071	4.907
2	4.249	4.167	4.296	4.442	4.512	4.563	4.416	4.079	4.591
3	5.122	5.136	4.824	5.092	4.620	4.599	4.653	5.057	4.964
4	3.534	3.593	3.454	3.566	3.765	3.683	3.602	3.586	3.471
5	3.292	2.993	2.928	3.088	3.013	3.122	3.020	3.108	2.958
6	3.159	3.679	3.796	3.516	3.429	3.447	3.564	3.570	3.764
7	3.036	2.952	3.028	3.189	2.963	2.912	3.176	3.139	2.974

Shading indicates lowest error

Table 6-4. Sum error

Based on the lowest sum error, the weights for the corresponding best levels are selected. Therefore, the best levels are 7, 7, 5, 5, 7, 7, 5, 5, 5 which produced the maximum output with zero error.

The second iteration was done merely to demonstrate that the error value consistently reduced with the number of iterations. As explained in Chapter 4 in section 4.2.2, the levels are modified using the best levels obtained from the first iteration.

Table 6-5 shows the errors after the second iteration. It can be seen that for the same experiment number 18, the error reduced to zero producing the maximum output. Interestingly there are other experiments (experiment nos. 9, 14, 15 etc.), which also produced zero error. It can also be observed that for any experiment in iteration 2, the error is less when compared to iteration 1 for the same experiment.

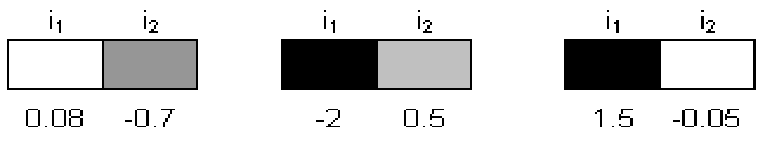
Table 6-5. Experiments showing errors after iteration 2

	NEURONS									Error	
	N1	N2	N3	N4	N5	N6	N7	N8	N9	**Iteration 1**	**Iteration 2**
Exp 1	0	5	6	0	1	2	2	5	1	**0.72**	**0.16**
Exp 2	0	6	4	4	2	7	3	0	7	**0.34**	**0.05**
Exp 3	0	4	2	3	3	4	6	3	2	**0.88**	**0.39**
Exp 4	0	7	3	6	4	0	1	7	3	**0.67**	**0.11**
Exp 5	0	3	7	1	0	5	0	6	4	**0.58**	**0.06**
Exp 6	0	0	1	5	6	1	7	4	0	**0.59**	**0.06**
Exp 7	0	2	0	7	7	6	4	2	5	**0.19**	**0.02**
Exp 8	0	1	5	2	5	3	5	1	6	**0.57**	**0.06**
Exp 9	7	5	4	3	4	5	7	2	6	**0.06**	**0.00**
Exp 10	7	6	6	6	3	1	0	1	5	**0.53**	**0.05**
Exp 11	7	4	3	0	2	6	5	6	0	**0.34**	**0.04**
Exp 12	7	7	2	4	1	3	4	4	4	**0.28**	**0.02**
Exp 13	7	3	1	7	5	2	3	3	3	**0.84**	**0.08**
Exp 14	7	0	7	2	7	7	2	7	2	**0.10**	**0.00**
Exp 15	7	2	5	1	6	4	1	5	7	**0.16**	**0.00**
Exp 16	7	1	0	5	0	0	6	0	1	**0.73**	**0.16**
Exp 17	5	5	2	1	7	1	5	0	3	**0.48**	**0.03**
Exp 18	5	6	3	5	5	5	4	5	2	**0.04**	**0.00**
Exp 19	5	4	6	7	0	3	7	7	7	**0.05**	**0.00**
Exp 20	5	7	4	2	6	6	0	3	1	**0.43**	**0.03**
Exp 21	5	3	0	0	3	7	1	4	6	**0.67**	**0.03**
Exp 22	5	0	5	4	4	2	6	6	5	**0.06**	**0.01**
Exp 23	5	2	7	3	1	0	3	1	0	**0.91**	**0.21**
Exp 24	5	1	1	6	2	4	2	2	4	**0.66**	**0.11**
Exp 25	2	5	3	7	6	7	6	1	4	**0.21**	**0.02**
Exp 26	2	6	2	2	0	2	1	2	0	**0.93**	**0.73**
Exp 27	2	4	4	1	5	0	2	4	5	**0.32**	**0.04**
Exp 28	2	7	6	5	7	4	3	6	6	**0.09**	**0.00**
Exp 29	2	3	5	3	2	1	4	7	1	**0.84**	**0.26**
Exp 30	2	0	0	6	1	5	5	3	7	**0.49**	**0.06**
Exp 31	2	2	1	0	4	3	0	0	2	**0.95**	**0.72**
Exp 32	2	1	7	4	3	6	7	5	3	**0.42**	**0.02**
Exp 33	3	5	7	5	2	3	1	3	5	**0.56**	**0.06**
Exp 34	3	6	1	1	1	6	6	7	6	**0.71**	**0.16**
Exp 35	3	4	0	2	4	1	3	5	4	**0.73**	**0.11**
Exp 36	3	7	5	7	3	5	2	0	0	**0.39**	**0.02**
Exp 37	3	3	6	4	6	0	5	2	2	**0.66**	**0.06**
Exp 38	3	0	4	0	0	4	4	1	3	**0.89**	**0.39**
Exp 39	3	2	2	6	5	7	7	6	1	**0.34**	**0.01**
Exp 40	3	1	3	3	7	2	0	4	7	**0.84**	**0.25**
Exp 41	4	5	1	2	3	0	4	6	7	**0.42**	**0.15**
Exp 42	4	6	7	7	4	4	5	4	1	**0.05**	**0.00**
Exp 43	4	4	5	5	1	7	0	2	3	**0.33**	**0.02**

Exp 44	4	7	0	1	2	2	7	1	2	**0.72**	**0.16**
Exp 45	4	3	4	6	7	3	6	5	0	**0.28**	**0.00**
Exp 46	4	0	6	3	5	6	1	0	4	**0.52**	**0.02**
Exp 47	4	2	3	4	0	1	2	3	6	**0.88**	**0.39**
Exp 48	4	1	2	0	6	5	3	7	5	**0.33**	**0.01**
Exp 49	6	5	0	4	5	4	0	7	0	**0.11**	**0.00**
Exp 50	6	6	5	0	7	0	7	3	4	**0.16**	**0.02**
Exp 51	6	4	7	6	6	2	4	0	6	**0.13**	**0.02**
Exp 52	6	7	1	3	0	7	5	5	5	**0.11**	**0.01**
Exp 53	6	3	2	5	4	6	2	1	7	**0.42**	**0.02**
Exp 54	6	0	3	1	3	3	3	2	1	**0.99**	**0.84**
Exp 55	6	2	6	2	2	5	6	4	3	**0.42**	**0.04**
Exp 56	6	1	4	7	1	1	1	6	2	**0.82**	**0.03**
Exp 57	1	5	5	6	0	6	3	4	2	**0.42**	**0.02**
Exp 58	1	6	0	3	6	3	2	6	3	**0.92**	**0.41**
Exp 59	1	4	1	4	7	5	1	1	1	**0.81**	**0.11**
Exp 60	1	7	7	0	5	1	6	2	7	**0.27**	**0.05**
Exp 61	1	3	3	2	1	4	7	0	5	**0.84**	**0.40**
Exp 62	1	0	2	7	2	0	0	5	6	**0.64**	**0.08**
Exp 63	1	2	4	5	3	2	5	7	4	**0.32**	**0.01**
Exp 64	1	1	6	1	4	7	4	3	0	**0.81**	**0.06**

6.3 Multiple Pattern Recognition

To ascertain the capabilities of the algorithm for training multiple patterns, the same network was tested using the following patterns:



Output =1 Output =0 Output =0

Network is trained with all three patterns and the total error is calculated which is given in Table 6-6. It can be observed that the lowest error is 1.05

Table 6-6. Error for multiple patterns

	NEURONS									
	N1	N2	N3	N4	N5	N6	N7	N8	N9	**Total Error**
Exp 1	0	5	6	0	1	2	2	5	1	**1.27**
Exp 2	0	6	4	4	2	7	3	0	7	**1.65**
Exp 3	0	4	2	3	3	4	6	3	2	**1.10**
Exp 4	0	7	3	6	4	0	1	7	3	**1.54**
Exp 5	0	3	7	1	0	5	0	6	4	**1.51**

Exp 6	0	0	1	5	6	1	7	4	0	**1.59**
Exp 7	0	2	0	7	7	6	4	2	5	**1.50**
Exp 8	0	1	5	2	5	3	5	1	6	**1.32**
Exp 9	7	5	4	3	4	5	7	2	6	**1.87**
Exp 10	7	6	6	6	3	1	0	1	5	**1.84**
Exp 11	7	4	3	0	2	6	5	6	0	**1.65**
Exp 12	7	7	2	4	1	3	4	4	4	**1.68**
Exp 13	7	3	1	7	5	2	3	3	3	**1.09**
Exp 14	7	0	7	2	7	7	2	7	2	**1.49**
Exp 15	7	2	5	1	6	4	1	5	7	**1.83**
Exp 16	7	1	0	5	0	0	6	0	1	**1.38**
Exp 17	5	5	2	1	7	1	5	0	3	**1.34**
Exp 18	5	6	3	5	5	5	4	5	2	**1.73**
Exp 19	5	4	6	7	0	3	7	7	7	**1.91**
Exp 20	5	7	4	2	6	6	0	3	1	**1.68**
Exp 21	5	3	0	0	3	7	1	4	6	**1.45**
Exp 22	5	0	5	4	4	2	6	6	5	**1.87**
Exp 23	5	2	7	3	1	0	3	1	0	**1.09**
Exp 24	5	1	1	6	2	4	2	2	4	**1.24**
Exp 25	2	5	3	7	6	7	6	1	4	**1.85**
Exp 26	2	6	2	2	0	2	1	2	0	**1.10**
Exp 27	2	4	4	1	5	0	2	4	5	**1.37**
Exp 28	2	7	6	5	7	4	3	6	6	**1.94**
Exp 29	2	3	5	3	2	1	4	7	1	**1.09**
Exp 30	2	0	0	6	1	5	5	3	7	**1.43**
Exp 31	2	2	1	0	4	3	0	0	2	**1.05**
Exp 32	2	1	7	4	3	6	7	5	3	**1.58**
Exp 33	3	5	7	5	2	3	1	3	5	**1.12**
Exp 34	3	6	1	1	1	6	6	7	6	**1.80**
Exp 35	3	4	0	2	4	1	3	5	4	**1.21**
Exp 36	3	7	5	7	3	5	2	0	0	**1.39**
Exp 37	3	3	6	4	6	0	5	2	2	**1.24**
Exp 38	3	0	4	0	0	4	4	1	3	**1.27**
Exp 39	3	2	2	6	5	7	7	6	1	**1.65**
Exp 40	3	1	3	3	7	2	0	4	7	**1.17**
Exp 41	4	5	1	2	3	0	4	6	7	**1.58**
Exp 42	4	6	7	7	4	4	5	4	1	**1.94**
Exp 43	4	4	5	5	1	7	0	2	3	**1.46**
Exp 44	4	7	0	1	2	2	7	1	2	**1.27**
Exp 45	4	3	4	6	7	3	6	5	0	**1.73**
Exp 46	4	0	6	3	5	6	1	0	4	**1.66**
Exp 47	4	2	3	4	0	1	2	3	6	**1.19**
Exp 48	4	1	2	0	6	5	3	7	5	**1.55**
Exp 49	6	5	0	4	5	4	0	7	0	**1.73**
Exp 50	6	6	5	0	7	0	7	3	4	**1.91**
Exp 51	6	4	7	6	6	2	4	0	6	**1.92**
Exp 52	6	7	1	3	0	7	5	5	5	**1.73**
Exp 53	6	3	2	5	4	6	2	1	7	**1.58**
Exp 54	6	0	3	1	3	3	3	2	1	**1.13**
Exp 55	6	2	6	2	2	5	6	4	3	**1.49**
Exp 56	6	1	4	7	1	1	1	6	2	**1.75**
Exp 57	1	5	5	6	0	6	3	4	2	**1.58**

Exp 58	1	6	0	3	6	3	2	6	3	**1.50**
Exp 59	1	4	1	4	7	5	1	1	1	**1.68**
Exp 60	1	7	7	0	5	1	6	2	7	**1.79**
Exp 61	1	3	3	2	1	4	7	0	5	**1.17**
Exp 62	1	0	2	7	2	0	0	5	6	**1.43**
Exp 63	1	2	4	5	3	2	5	7	4	**1.37**
Exp 64	1	1	6	1	4	7	4	3	0	**1.68**

As explained before, sum error is calculated, and based on the lowest sum error the weights for the corresponding best levels are selected. The best levels produced an error of 0.97, which is less than the error produced by the table of experiments.

This method of training gives a better error reduction than applying the standard method. The error reduction is generally good, but it is not always the lowest which is theoretically possible. The reason for this is interactions between the neurons are replacing interactions between the layers as a problem. The advantage with this method is custom-made OAs can be used to accommodate different size of networks.

Chapter 7

Using New Methods in Learning Non-linear Functions

7.1 Introduction to Chapter

In the previous chapters, it has been demonstrated that Taguchi Methods may be applied to train a single neuron, single layer networks and multi-layer networks successfully. The Neuron-By-Neuron training method and the polynomial method are particularly effective. These methods give the optimum weights in all tests.

In this chapter, attention is drawn to the following areas:

- A comparison is made between the Neuron-By-Neuron training method and the standard back-propagation algorithm in terms of training speed.
- To fully illustrate its capabilities, one of the new methods was tested with non-linear training functions - Sigmoid, Reverse Sigmoid and Gaussian (bell curve). A comparison is made between the theoretical (expected) and actual outputs of the function.

7.2 Comparison with the Back Propagation Algorithm

It is difficult to assess different training algorithms in terms of their comparative speed, as this depends on many factors. For example, the complexity of the problem, the number of patterns being trained, the number of weights in the network, the error goal, and whether the network is being used for pattern recognition (discriminant analysis) or function approximation (regression), etc.

To have a like-with-like comparison of a known pattern recognition problem, a multi-layer network and sample test patterns were chosen. The network is shown in Figure 7-1. Two tests have been conducted. The first test (Test 1) uses two patterns, which are shown in Figure 7-2.

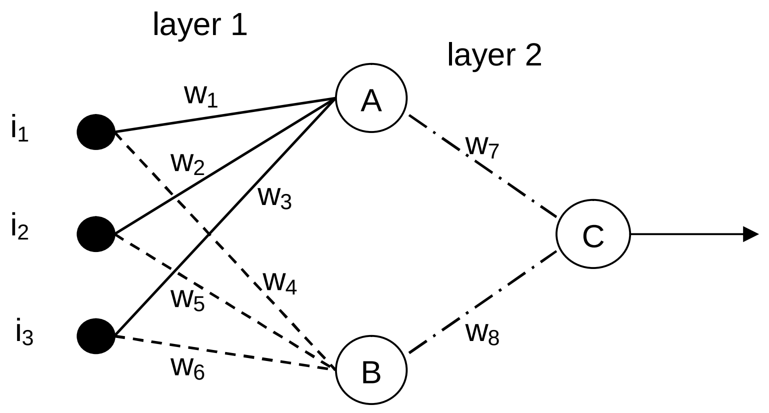


Figure 7-1. Network for training cycles comparison

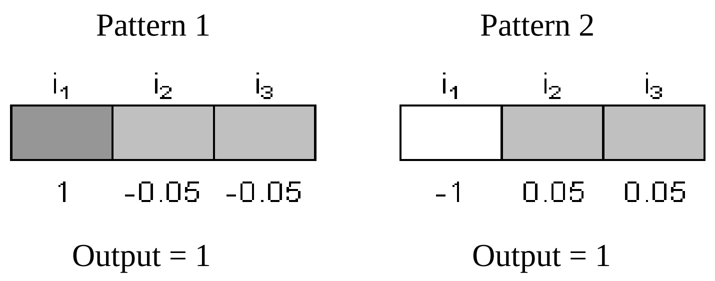


Figure 7-2. Test 1 patterns

The network was trained using the Neuron-By-Neuron Method of Training, demonstrated in Chapter 5. The error after just one pass was 0.538. The same network with same patterns was trained using Back-Propagation algorithm. To reach the closest error of 0.53798, the BP algorithm took 293 iterations (training cycles).

To get a better picture, both the methods of training were looked at more closely in terms of calculations. Table 7-1 gives the arithmetical operations involved for one iteration, for each method.

	Back Propagation	Taguchi Methods
Adds/Subtracts	44	252
Multiplies/Divides	70	264
Exponentiations	6	72

Table 7-1. Comparative data for one iteration

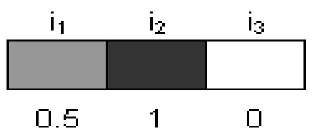
To reach the same error of 0.53798, BP algorithm took 293 iterations. Hence, the arithmetic operations for BP shown in Table 7-1 were multiplied by 293, and this is shown in Table 7-2.

	Back Propagation	Taguchi Methods
Adds/Subtracts	12892	252
Multiplies/Divides	20510	264
Exponentiations	1758	72

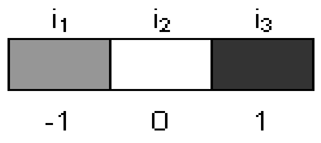
Table 7-2. Arithmetic operations required to reach the same error

Therefore, for this example, the Neuron-By-Neuron training method was faster than BP by a factor of 60.

The same network was trained with another set of patterns, which are given in Figure 7-3.



Output = 1



Output = 1

Figure 7-3. Test 2 patterns

Using the Neuron-By-Neuron training method, the error after just one pass was 0.370. Again, this was trained using the Back-Propagation algorithm. To reach approximately the same error of 0.37016, the BP algorithm took 514 generations.

As explained before, Table 7-3 shows the comparative data for this example.

	Back Propagation	Taguchi Methods
Adds/Subtracts	22616	252
Multiplies/Divides	35980	264
Exponentiations	3084	72

Table 7-3. Comparative data for Test 2

Hence, in this example, the Neuron-By-Neuron training method was faster than BP by a factor of 100.

The above examples illustrate the range of training speed improvements possible in a particular set of small pattern recognition problems. For the size of networks tested (illustrated in the previous figures), the method is at least 60 times faster than BP. Note, however, that this improvement varies with different sized networks.

7.3 Learning Non-linear Functions

To test the effectiveness of the training methods developed it was decided to test the algorithm’s effectiveness in learning some non-linear functions. These were selected because they might be encountered as compensation functions in non-linear control situations (Nijmeijer et al 1990). These were, the Sigmoidal Function (equation 1), its inverse and the bell (Gaussian) curve (equation 2).

Equation 1
$$f(x) = \frac{1}{1+e^{-\mu x}}$$

Equation 2
$$f(x) = \frac{e^{-x^2/2}}{\sqrt{2\pi}}$$

Where μ is an arbitrary constant which governs the shape of the curve.

To do this, a nine neuron hidden layer, 2 input and one output network (MacLeod et al 2003) was constructed. One of the new training methods, ‘Coding the state of Neuron’ was used to train this network. However, any other successful new method could have been used since they showed better results when used for training.

To train this network, one input was held at a constant bias level of 1, the other was subject to a linear ramp of one-unit steps, which is illustrated in Figure 7-4. The network was trained with a single pass. Results are shown below in figure 7-5 for Sigmoid Function, Reverse Sigmoid and Gaussian transfer functions. Thus, the network successfully learns non-linear mappings.

Figure 7-4. Inputs for the network

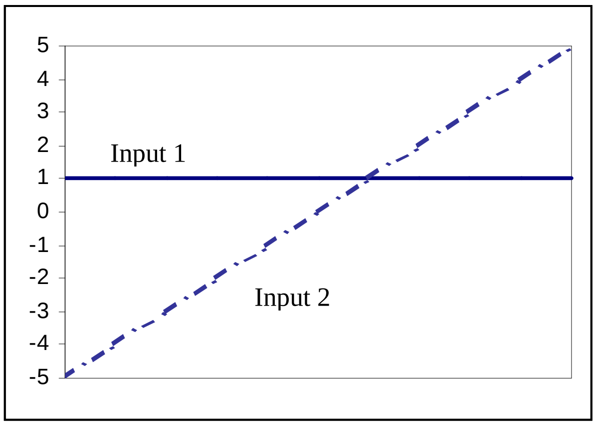
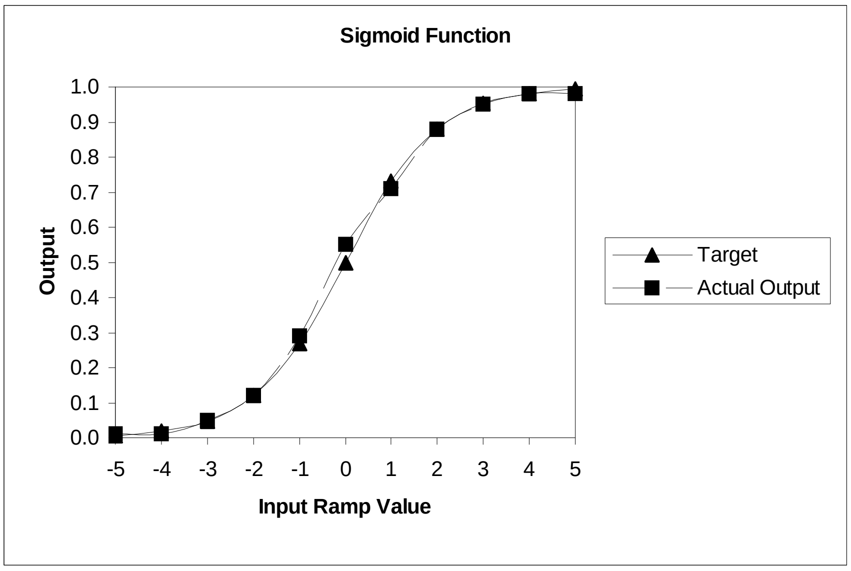
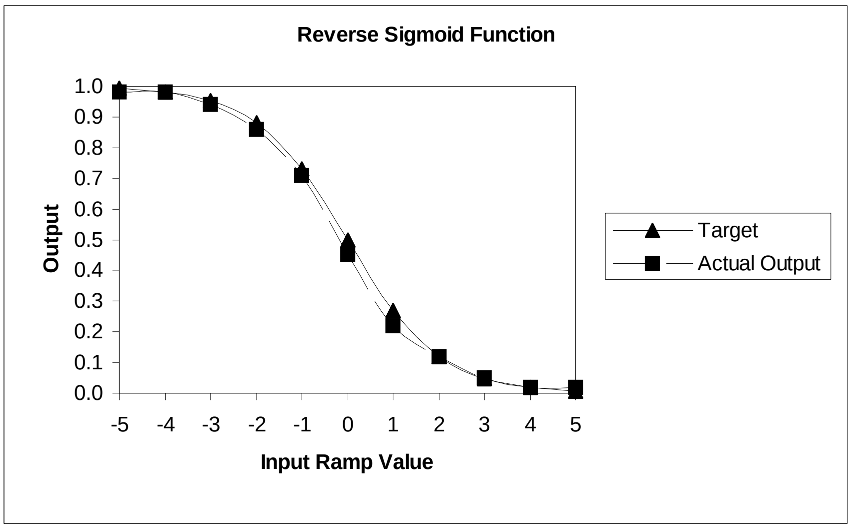
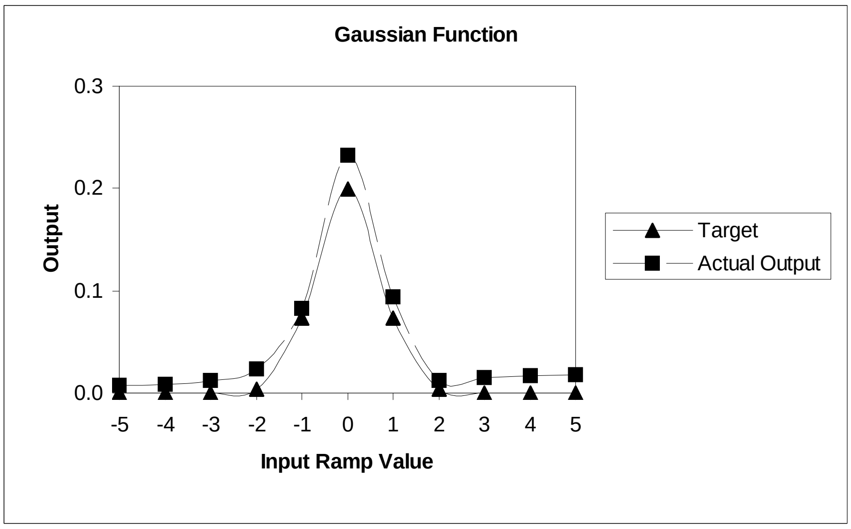


Figure 7-5. Taguchi learning of non-linear compensation functions







In this chapter, a comparison was made between the Neuron-By-Neuron training method and the standard back-propagation algorithm in terms of training speed. It was shown that the new method trains the network at a much faster rate than BP. To explore the full capabilities of the method, it was tested on non-linear training functions - sigmoid, reverse sigmoid and Gaussian (bell curve). Comparison is made between theoretical and actual outputs of the function. The network successfully learns the non-linear mappings. This was demonstrated by using ‘Coding the state of Neuron’ training method, and it is also possible to produce the same results using the other new methods developed.

Chapter 8
Conclusions

8.1 Introduction to Chapter

This final chapter presents the conclusions of the project. The objectives, as specified in Chapter 1, are reviewed with reference to the work presented in the previous chapters. A brief discussion of original contributions to the art and suggestions for further work are made. Finally, some concluding remarks about the project as a whole are given.

8.2 The Project Objectives Revisited

The project objectives, as defined in Chapter 1, are listed below:

1. Undertake a study of relevant literature
2. Train a simple neuron using Taguchi Methods
3. Investigate new methods of training ANNs using Taguchi Methods
4. Investigate use of custom-made OAs for training ANNs
5. Use the new methods developed to learn non-linear functions
6. Comparison against other published work

The following sections look at each of these objectives in turn and consider how well they have been achieved.

8.2.1 Undertake a study of relevant literature

Literature search was done in a wide range of topics including Taguchi Methods, Orthogonal Arrays, ANN training and other training methods. This was done extensively at the beginning of research and continued throughout the duration of the project, at a slightly lower level. The results of these are given in Chapter 3, although references are made to appropriate material throughout the thesis.

8.2.2 Train a simple neuron using Taguchi Methods

The primary aim here was to investigate the use of Taguchi Methods to train a simple neuron, then progress to single layer networks, and finally to multi-layer networks. Taguchi Methods were successfully applied to the training of a simple neuron and also to single layer networks. However, when they were tried on multi-layer networks, the training failed due to interaction between the interlayer weights. These results are discussed in Chapter 4.

8.2.3 Investigate new methods of training ANNs using Taguchi Methods

A number of alternate training strategies were considered to train multi-layer networks using Taguchi Methods. These methods mainly focused on overcoming the interaction problem. A number of training strategies were developed to train multi-layer networks. These were ‘Neuron-By-Neuron Training Method’ and ‘Polynomial (power series) neurons’. Taguchi Methods were successfully demonstrated for neural network training using these techniques. Detailed explanations of the techniques are given in Chapter 5.

8.2.4 Investigate use of custom-made OAs for training ANNs

It had been highlighted that selecting a suitable OA table for a given problem may be difficult under some circumstances. Therefore, another new method of training was also developed that uses custom-made OA tables. Chapter 6 discusses this method ‘Coding the state of Neuron’ training.

8.2.5 Use the new methods developed to learn non-linear functions

The ‘Coding the state of Neuron’ training method was successfully demonstrated using non-linear training functions - Sigmoid, Reverse Sigmoid and Gaussian (bell curve). A comparison is made between the theoretical and actual outputs of these functions. Chapter 7 discusses the results obtained.

8.2.6 Comparison against other published work

A comparison is made between the method and the standard back-propagation algorithm in terms of their training speed. When compared with Back-Propagation, the method reduces the network training time significantly. This is discussed in Chapter 7.

8.3 Original Contributions

When assessing any research project, an inevitable question concerns the contribution to new knowledge made by the researchers and their work. This project has several original aspects to it, all of which are a product of the work. These are:

- The origination and testing of an innovative 'Neuron-By-Neuron Method of Training', which is based on Taguchi Methods, was developed to train Artificial Neural Networks.
- To train Polynomial (power series) neurons, another unique training method was developed using Taguchi Methods. The network error reduces with each increasing input power as the approximation becomes more accurate.
- A unique method of 'Coding the state of Neuron' training, also based on Taguchi Methods, was developed and tested. This method uses custom-made Orthogonal Array tables.
- When compared with traditional algorithms like Back-Propagation, the new method reduces the network training time significantly. It was also demonstrated in learning some non-linear functions.

Although the idea of using Taguchi Methods to train ANNs is not new, the training strategies, which are listed above, are original and none had been tested in depth until this project. These have been proved effective as a means of training multi-layer ANNs.

8.4 Suggestions for Further Work

There are two main areas in which further work could be carried out to extend this project.

The first area is to explore ways of improving the training methods further. With the new methods, it may not always be possible to reach the desired error level after just one pass (iteration). Hence, the new methods may be used to reduce the error level significantly after just one pass, and then a Gradient-Descent algorithm such as Back-Propagation (which allows the weights to change in small increments) may be used to 'fine-tune' the weights, so that the desired error level may be achieved. However, Taguchi Methods can also be used iteratively, which is demonstrated in Chapter 4.

It would be possible to combine Polynomial neurons with the Neuron-By-Neuron method of training, so that each polynomial neuron can be trained, and then another added. It is also possible to iterate this method to reduce the error to the desired level.

Therefore, further work is suggested in this area to explore these possibilities.

The other area is, to implement the algorithm on a simple control system, for example, controlling a DC motor. Although it is highly desirable to train the ANN at a faster rate, irrespective of the application it is being used, it is very critical to train such networks very quickly when used in certain Control Systems, the reason being that these may be used as 'disaster' control. One well-known example of the application of the scheme is the failure of an aerodynamic surface in an aircraft – it may be possible to use this technique to bring such a potentially dangerous system 'back under control'. Therefore, further work is suggested to implement the new training methods on simple control systems.

8.5 Concluding Remarks

The project has been successful in that all the objectives have been met, and that the scheme worked as expected.

The methods developed are powerful and useful for reducing the network error significantly. Alternate training strategies have been developed to use in different situations.

This work now joins a body of other research describing all training algorithms in training ANNs.

References

[1] Altun, H., Curtis, K.M., 1997. An improved neural network learning. *Proceedings of the Fourth IEEE International Conference on Electronics, Circuits and Systems*, vol.1 pt.1, p 29-33

[2] Chang, C.C., Chang, T.Y.P., Xu, Y.G., To, W.M., Jan 31 2003, 2002. Selection of training samples for model updating using neural networks. *Journal of Sound and Vibration*, v 249, n 5, p 867-883

[3] Dey, A., 1985. *Orthogonal Fractional Designs*. Halstead Press.

[4] Dror, G., 1995. *Training Neural Networks Using Taguchi Methods*, Thesis (MSc). The Robert Gordon University.

[5] Dubrovin, V., Subbotin, S., 2002. The quick method of neural network training. *Modern Problems of Radio Engineering, Telecommunications and Computer Science*, IEEE Cat. No.02EX542, p 266-7

[6] Haykin, S., 1999. *Neural Networks: A Comprehensive Foundation*, Prentice-Hall.

[7] Kawata, H., Taguchi, A., Sone, M., Dec 1998. Pattern error equality learning of neural network with the genetic algorithm. *Transactions of the Institute of Electronics, Information and Communication Engineers.* D-II, v J81D-II, n 12, p 2841-9

[8] Khanna, T., 1990. *Foundation of Neural Networks*, Addison Wesley, London

[9] Khaw, J.F.C., Lim, B.S., Lim, L.E.N., April 1995. Optimal design of neural networks using the Taguchi method. *Neurocomputing*, v 7, n 3, p 225-45

[10] Looney, C.G., Tacker, E.C., 1992. Neural learning and computing with multiple weight sets. *Control and Computers*, v 20, n 3, p 97-102

[11] MacLeod C., Dror G., Maxwell G., 1999. Training Artificial Neural Networks using Taguchi Methods. *Artificial Intelligence Review* 13, 177-184,

[12] MacLeod, C., Maxwell, G., 2003. An Introduction to Neural Nets - Practical Neural Networks, Part1. *Elektor Electronics*. January.

[13] Maxwell G., Macleod C., 2002. Taguchi Methods for Pattern Recognition, Control and Evolutionary Applications, *ICONIP*, vol 1, p301-305

[14] McMinn, D., 2002. *Using Evolutionary Artificial Neural Networks to design hierachial animat nervous systems*, Thesis (PhD). The Robert Gordon University.

[15] Ming-Der, J., Chyuan-Du, L., Jen-Ting, W., May 2005. Design and development of artificial neural networks for depositing powders in coating treatment. *Applied Surface Science*, Volume 245, Issues 1-4, 30, Pages 290-303

[16] Mo-Chung, L., Young, M.S., Bon, H., Ma, R.P., Chii-Maw, U., 1999. The application Taguchi quality engineering method on backpropagation neural network. *Optical memory & Neural Networks*, v 8, n 2, p 91-103

[17] Nijmeijer, H., Van der Schaft, A J., Van der schaft, A., 1990. *Nonlinear dynamical control systems*, Springer-Verlag Berlin

[18] Owen, A., 1992. Orthogonal Arrays for Computer Experiments, Integration and Visualisation. *Statistica Sinica 2*, pp439-452

[19] Owen, A., 2004. *StatLib-Designs.* Available from: http://lib.stat.cmu.edu/designs/owen.html [Accessed 2 March 2003]

[20] Packianather, M.S., Drake, P.R., Rowlands, H., Nov-Dec.2000. Optimizing the parameters of multilayered feedforward neural networks through Taguchi Design of Experiments. *Quality and Reliability Engineering International*, v 16, n 6, p 461-73

[21] Peterson, G.E., St.Clair, D.C., Aylward, S.R., Bond, W.E., July 1995. Using Taguchi's method of experimental design to control errors in layered perceptrons. *IEEE Transactions on Neural Networks*, v 6, n 4, p 949-61

[22] Ranjit, K.R., 1990. *A Primer on the Taguchi Method*. Van Nostrand Reinhold.

[23] Ross, P.J., 1988. *Taguchi Techniques for Quality Engineering*. McGraw-Hill.

[24] Rowlands, H., Packianather, M.S., Oztemel, E., 1996. Using artificial neural networks for experimental design in off-line quality control. *Journal of Systems Engineering*, v 6, n 1, p 46-59

[25] Sloane, N. J. A., 2004. *A Library of Orthogonal Arrays*. Available from: http://www.research.att.com/~njas/oadir/ [Accessed 2 June 2004]

[26] Stoica, A., Blosiu, J., 1997. Neural Learning using Orthogonal Arrays, *Advances in Intelligent Systems*, IOS Press

[27] Su, C., Chang, H., Dec 2000. Optimization of parameter design: An intelligent approach using neural network and simulated annealing. *International Journal of Systems Science*, v 31, n 12, p 1543-1549

[28] Van Laarhoven, P.J.M., 1988. *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers.

[29] Viswanathan A., Macleod C., Maxwell G., Kalidindi S. 2005. Orhtogonal Arrays for Training Artificial Neural Networks - Overcoming Interaction Problems, *ICANN*, Springer-Verlag Berlin. p 103-108

[30] Wasserman, P.D., 1989. *Neural Computing: Theory and Practice*, Van Nostrand Reinhold.

[31] Yang, S.M., Lee, G.S., Sep 1999. Neural network design by using Taguchi method. *Transactions of the ASME. Journal of Dynamic Systems, Measurement and Control*, v 121, n 3, p 560-3

[32] Yamada, K., Iigima, N., Akima, Y., Taguchi, A., Sone, M., 1994. A fast training algorithm for neural networks. *Proceedings of the IASTED International Conference. Modelling and Simulation*. p 340-2

[33] Young-Sang, K., Bong-Jin, Y., April 2004. Robust design of multilayer feeedforward neural networks: an experimental approach. *Engineering Applications of Artificial Intelligence*, v 17, n 3, p 249-63

[34] Zhang, S.J., Jing, Z.L., Li, J.X., Sep 2004. Fast learning high-order neural networks for pattern recognition. *Electronics Letters*, v 40, n 19, 16, p 1207-8

Appendix one contained a paper summarising the work, presented at the ICANN conference. This is not available in PDF format for technical reasons, but may be ordered from the usual sources (for example, the publisher or the British Library). The full reference is:

A Viswanathan, C MacLeod, G Maxwell, S Kalidindi, *Training Neural Networks using Taguchi Methods: Overcoming Interaction Problems*, ICANN 05, Warsaw Poland, Part II, p 103 – 108.

APPENDICES

Appendix - A2

Methods Used to Obtain Results

All the experiments were performed using C++ programming and in MS Excel using Macro programming.

Training polynomial neurons, Taguchi iterative training methods and Training Back-Propagation networks have been coded in C++. The other training methods using Taguchi Methods and the full factorial table results have been done in MS Excel using Macro programming.

The levels in Orthogonal Array table for the experiment was used with quantised weights for the first iteration. These have been modified based on Taguchi Methods calculations, and the modified weights corresponding to the levels were used for the subsequent iterations. These are performed automatically by a C++ program.

Training a network using Back-Propagation algorithm was done using C++ program. One can train the network until the desired error level is reached.

Coding the state of each neuron training method was developed using the custom-made OA from Owen's website. The OA table was transformed into spreadsheet and the training was done using Macros. The program performs each of the experiment from the OA table and calculates the network error. The program updates the weights corresponding to the levels for further iterations. The other training methods, neuron-by-neuron training and layer-by-layer training were done in a similar way.

Coding of polynomial neurons training method was done in C++. The series for different orders like squared term, cubed term etc. have been coded. The program is also capable of computing network error with factorials.

Full factorial experiments

	W1	W2	W3	W4	W5	W6	W7	W8	**Total Error**
Exp 1	-1	-1	-1	-1	-1	-1	-1	-1	1.446
Exp 2	1	-1	-1	-1	-1	-1	-1	-1	1.462
Exp 3	-1	1	-1	-1	-1	-1	-1	-1	1.445
Exp 4	1	1	-1	-1	-1	-1	-1	-1	1.462
Exp 5	-1	-1	1	-1	-1	-1	-1	-1	1.445
Exp 6	1	-1	1	-1	-1	-1	-1	-1	1.462
Exp 7	-1	1	1	-1	-1	-1	-1	-1	1.443
Exp 8	1	1	1	-1	-1	-1	-1	-1	1.462
Exp 9	-1	-1	-1	1	-1	-1	-1	-1	1.462
Exp 10	1	-1	-1	1	-1	-1	-1	-1	1.440
Exp 11	-1	1	-1	1	-1	-1	-1	-1	1.462
Exp 12	1	1	-1	1	-1	-1	-1	-1	1.442
Exp 13	-1	-1	1	1	-1	-1	-1	-1	1.462
Exp 14	1	-1	1	1	-1	-1	-1	-1	1.442
Exp 15	-1	1	1	1	-1	-1	-1	-1	1.462
Exp 16	1	1	1	1	-1	-1	-1	-1	1.443
Exp 17	-1	-1	-1	-1	1	-1	-1	-1	1.445
Exp 18	1	-1	-1	-1	1	-1	-1	-1	1.462
Exp 19	-1	1	-1	-1	1	-1	-1	-1	1.443
Exp 20	1	1	-1	-1	1	-1	-1	-1	1.462
Exp 21	-1	-1	1	-1	1	-1	-1	-1	1.443
Exp 22	1	-1	1	-1	1	-1	-1	-1	1.462
Exp 23	-1	1	1	-1	1	-1	-1	-1	1.442
Exp 24	1	1	1	-1	1	-1	-1	-1	1.462
Exp 25	-1	-1	-1	1	1	-1	-1	-1	1.462
Exp 26	1	-1	-1	1	1	-1	-1	-1	1.442
Exp 27	-1	1	-1	1	1	-1	-1	-1	1.462
Exp 28	1	1	-1	1	1	-1	-1	-1	1.443
Exp 29	-1	-1	1	1	1	-1	-1	-1	1.462
Exp 30	1	-1	1	1	1	-1	-1	-1	1.443
Exp 31	-1	1	1	1	1	-1	-1	-1	1.462
Exp 32	1	1	1	1	1	-1	-1	-1	1.445
Exp 33	-1	-1	-1	-1	-1	1	-1	-1	1.445
Exp 34	1	-1	-1	-1	-1	1	-1	-1	1.462
Exp 35	-1	1	-1	-1	-1	1	-1	-1	1.443
Exp 36	1	1	-1	-1	-1	1	-1	-1	1.462
Exp 37	-1	-1	1	-1	-1	1	-1	-1	1.443
Exp 38	1	-1	1	-1	-1	1	-1	-1	1.462
Exp 39	-1	1	1	-1	-1	1	-1	-1	1.442
Exp 40	1	1	1	-1	-1	1	-1	-1	1.462
Exp 41	-1	-1	-1	1	-1	1	-1	-1	1.462
Exp 42	1	-1	-1	1	-1	1	-1	-1	1.442
Exp 43	-1	1	-1	1	-1	1	-1	-1	1.462
Exp 44	1	1	-1	1	-1	1	-1	-1	1.443
Exp 45	-1	-1	1	1	-1	1	-1	-1	1.462
Exp 46	1	-1	1	1	-1	1	-1	-1	1.443
Exp 47	-1	1	1	1	-1	1	-1	-1	1.462
Exp 48	1	1	1	1	-1	1	-1	-1	1.445
Exp 49	-1	-1	-1	-1	1	1	-1	-1	1.443
Exp 50	1	-1	-1	-1	1	1	-1	-1	1.462

Exp 51	-1	1	-1	-1	1	1	-1	-1	1.442
Exp 52	1	1	-1	-1	1	1	-1	-1	1.462
Exp 53	-1	-1	1	-1	1	1	-1	-1	1.442
Exp 54	1	-1	1	-1	1	1	-1	-1	1.462
Exp 55	-1	1	1	-1	1	1	-1	-1	1.440
Exp 56	1	1	1	-1	1	1	-1	-1	1.462
Exp 57	-1	-1	-1	1	1	1	-1	-1	1.462
Exp 58	1	-1	-1	1	1	1	-1	-1	1.443
Exp 59	-1	1	-1	1	1	1	-1	-1	1.462
Exp 60	1	1	-1	1	1	1	-1	-1	1.445
Exp 61	-1	-1	1	1	1	1	-1	-1	1.462
Exp 62	1	-1	1	1	1	1	-1	-1	1.445
Exp 63	-1	1	1	1	1	1	-1	-1	1.462
Exp 64	1	1	1	1	1	1	-1	-1	1.446
Exp 65	-1	-1	-1	-1	-1	-1	1	-1	1.000
Exp 66	1	-1	-1	-1	-1	-1	1	-1	1.000
Exp 67	-1	1	-1	-1	-1	-1	1	-1	1.000
Exp 68	1	1	-1	-1	-1	-1	1	-1	1.000
Exp 69	-1	-1	1	-1	-1	-1	1	-1	1.000
Exp 70	1	-1	1	-1	-1	-1	1	-1	1.000
Exp 71	-1	1	1	-1	-1	-1	1	-1	1.000
Exp 72	1	1	1	-1	-1	-1	1	-1	1.000
Exp 73	-1	-1	-1	1	-1	-1	1	-1	1.000
Exp 74	1	-1	-1	1	-1	-1	1	-1	1.000
Exp 75	-1	1	-1	1	-1	-1	1	-1	1.000
Exp 76	1	1	-1	1	-1	-1	1	-1	1.000
Exp 77	-1	-1	1	1	-1	-1	1	-1	1.000
Exp 78	1	-1	1	1	-1	-1	1	-1	1.000
Exp 79	-1	1	1	1	-1	-1	1	-1	1.000
Exp 80	1	1	1	1	-1	-1	1	-1	1.000
Exp 81	-1	-1	-1	-1	1	-1	1	-1	1.000
Exp 82	1	-1	-1	-1	1	-1	1	-1	1.000
Exp 83	-1	1	-1	-1	1	-1	1	-1	1.000
Exp 84	1	1	-1	-1	1	-1	1	-1	1.000
Exp 85	-1	-1	1	-1	1	-1	1	-1	1.000
Exp 86	1	-1	1	-1	1	-1	1	-1	1.000
Exp 87	-1	1	1	-1	1	-1	1	-1	1.000
Exp 88	1	1	1	-1	1	-1	1	-1	1.000
Exp 89	-1	-1	-1	1	1	-1	1	-1	1.000
Exp 90	1	-1	-1	1	1	-1	1	-1	1.000
Exp 91	-1	1	-1	1	1	-1	1	-1	1.000
Exp 92	1	1	-1	1	1	-1	1	-1	1.000
Exp 93	-1	-1	1	1	1	-1	1	-1	1.000
Exp 94	1	-1	1	1	1	-1	1	-1	1.000
Exp 95	-1	1	1	1	1	-1	1	-1	1.000
Exp 96	1	1	1	1	1	-1	1	-1	1.000
Exp 97	-1	-1	-1	-1	-1	1	1	-1	1.000
Exp 98	1	-1	-1	-1	-1	1	1	-1	1.000
Exp 99	-1	1	-1	-1	-1	1	1	-1	1.000
Exp 100	1	1	-1	-1	-1	1	1	-1	1.000
Exp 101	-1	-1	1	-1	-1	1	1	-1	1.000
Exp 102	1	-1	1	-1	-1	1	1	-1	1.000
Exp 103	-1	1	1	-1	-1	1	1	-1	1.000
Exp 104	1	1	1	-1	-1	1	1	-1	1.000
Exp 105	-1	-1	-1	1	-1	1	1	-1	1.000

Exp 106	1	-1	-1	1	-1	1	1	-1	1.000
Exp 107	-1	1	-1	1	-1	1	1	-1	1.000
Exp 108	1	1	-1	1	-1	1	1	-1	1.000
Exp 109	-1	-1	1	1	-1	1	1	-1	1.000
Exp 110	1	-1	1	1	-1	1	1	-1	1.000
Exp 111	-1	1	1	1	-1	1	1	-1	1.000
Exp 112	1	1	1	1	-1	1	1	-1	1.000
Exp 113	-1	-1	-1	-1	1	1	1	-1	1.000
Exp 114	1	-1	-1	-1	1	1	1	-1	1.000
Exp 115	-1	1	-1	-1	1	1	1	-1	1.000
Exp 116	1	1	-1	-1	1	1	1	-1	1.000
Exp 117	-1	-1	1	-1	1	1	1	-1	1.000
Exp 118	1	-1	1	-1	1	1	1	-1	1.000
Exp 119	-1	1	1	-1	1	1	1	-1	1.000
Exp 120	1	1	1	-1	1	1	1	-1	1.000
Exp 121	-1	-1	-1	1	1	1	1	-1	1.000
Exp 122	1	-1	-1	1	1	1	1	-1	1.000
Exp 123	-1	1	-1	1	1	1	1	-1	1.000
Exp 124	1	1	-1	1	1	1	1	-1	1.000
Exp 125	-1	-1	1	1	1	1	1	-1	1.000
Exp 126	1	-1	1	1	1	1	1	-1	1.000
Exp 127	-1	1	1	1	1	1	1	-1	1.000
Exp 128	1	1	1	1	1	1	1	-1	1.000
Exp 129	-1	-1	-1	-1	-1	-1	-1	1	1.000
Exp 130	1	-1	-1	-1	-1	-1	-1	1	1.000
Exp 131	-1	1	-1	-1	-1	-1	-1	1	1.000
Exp 132	1	1	-1	-1	-1	-1	-1	1	1.000
Exp 133	-1	-1	1	-1	-1	-1	-1	1	1.000
Exp 134	1	-1	1	-1	-1	-1	-1	1	1.000
Exp 135	-1	1	1	-1	-1	-1	-1	1	1.000
Exp 136	1	1	1	-1	-1	-1	-1	1	1.000
Exp 137	-1	-1	-1	1	-1	-1	-1	1	1.000
Exp 138	1	-1	-1	1	-1	-1	-1	1	1.000
Exp 139	-1	1	-1	1	-1	-1	-1	1	1.000
Exp 140	1	1	-1	1	-1	-1	-1	1	1.000
Exp 141	-1	-1	1	1	-1	-1	-1	1	1.000
Exp 142	1	-1	1	1	-1	-1	-1	1	1.000
Exp 143	-1	1	1	1	-1	-1	-1	1	1.000
Exp 144	1	1	1	1	-1	-1	-1	1	1.000
Exp 145	-1	-1	-1	-1	1	-1	-1	1	1.000
Exp 146	1	-1	-1	-1	1	-1	-1	1	1.000
Exp 147	-1	1	-1	-1	1	-1	-1	1	1.000
Exp 148	1	1	-1	-1	1	-1	-1	1	1.000
Exp 149	-1	-1	1	-1	1	-1	-1	1	1.000
Exp 150	1	-1	1	-1	1	-1	-1	1	1.000
Exp 151	-1	1	1	-1	1	-1	-1	1	1.000
Exp 152	1	1	1	-1	1	-1	-1	1	1.000
Exp 153	-1	-1	-1	1	1	-1	-1	1	1.000
Exp 154	1	-1	-1	1	1	-1	-1	1	1.000
Exp 155	-1	1	-1	1	1	-1	-1	1	1.000
Exp 156	1	1	-1	1	1	-1	-1	1	1.000
Exp 157	-1	-1	1	1	1	-1	-1	1	1.000
Exp 158	1	-1	1	1	1	-1	-1	1	1.000
Exp 159	-1	1	1	1	1	-1	-1	1	1.000
Exp 160	1	1	1	1	1	-1	-1	1	1.000

Exp 161	-1	-1	-1	-1	-1	1	-1	1	1.000
Exp 162	1	-1	-1	-1	-1	1	-1	1	1.000
Exp 163	-1	1	-1	-1	-1	1	-1	1	1.000
Exp 164	1	1	-1	-1	-1	1	-1	1	1.000
Exp 165	-1	-1	1	-1	-1	1	-1	1	1.000
Exp 166	1	-1	1	-1	-1	1	-1	1	1.000
Exp 167	-1	1	1	-1	-1	1	-1	1	1.000
Exp 168	1	1	1	-1	-1	1	-1	1	1.000
Exp 169	-1	-1	-1	1	-1	1	-1	1	1.000
Exp 170	1	-1	-1	1	-1	1	-1	1	1.000
Exp 171	-1	1	-1	1	-1	1	-1	1	1.000
Exp 172	1	1	-1	1	-1	1	-1	1	1.000
Exp 173	-1	-1	1	1	-1	1	-1	1	1.000
Exp 174	1	-1	1	1	-1	1	-1	1	1.000
Exp 175	-1	1	1	1	-1	1	-1	1	1.000
Exp 176	1	1	1	1	-1	1	-1	1	1.000
Exp 177	-1	-1	-1	-1	1	1	-1	1	1.000
Exp 178	1	-1	-1	-1	1	1	-1	1	1.000
Exp 179	-1	1	-1	-1	1	1	-1	1	1.000
Exp 180	1	1	-1	-1	1	1	-1	1	1.000
Exp 181	-1	-1	1	-1	1	1	-1	1	1.000
Exp 182	1	-1	1	-1	1	1	-1	1	1.000
Exp 183	-1	1	1	-1	1	1	-1	1	1.000
Exp 184	1	1	1	-1	1	1	-1	1	1.000
Exp 185	-1	-1	-1	1	1	1	-1	1	1.000
Exp 186	1	-1	-1	1	1	1	-1	1	1.000
Exp 187	-1	1	-1	1	1	1	-1	1	1.000
Exp 188	1	1	-1	1	1	1	-1	1	1.000
Exp 189	-1	-1	1	1	1	1	-1	1	1.000
Exp 190	1	-1	1	1	1	1	-1	1	1.000
Exp 191	-1	1	1	1	1	1	-1	1	1.000
Exp 192	1	1	1	1	1	1	-1	1	1.000
Exp 193	-1	-1	-1	-1	-1	-1	1	1	0.554
Exp 194	1	-1	-1	-1	-1	-1	1	1	0.538
Exp 195	-1	1	-1	-1	-1	-1	1	1	0.555
Exp 196	1	1	-1	-1	-1	-1	1	1	0.538
Exp 197	-1	-1	1	-1	-1	-1	1	1	0.555
Exp 198	1	-1	1	-1	-1	-1	1	1	0.538
Exp 199	-1	1	1	-1	-1	-1	1	1	0.557
Exp 200	1	1	1	-1	-1	-1	1	1	0.538
Exp 201	-1	-1	-1	1	-1	-1	1	1	0.538
Exp 202	1	-1	-1	1	-1	-1	1	1	0.560
Exp 203	-1	1	-1	1	-1	-1	1	1	0.538
Exp 204	1	1	-1	1	-1	-1	1	1	0.558
Exp 205	-1	-1	1	1	-1	-1	1	1	0.538
Exp 206	1	-1	1	1	-1	-1	1	1	0.558
Exp 207	-1	1	1	1	-1	-1	1	1	0.538
Exp 208	1	1	1	1	-1	-1	1	1	0.557
Exp 209	-1	-1	-1	-1	1	-1	1	1	0.555
Exp 210	1	-1	-1	-1	1	-1	1	1	0.538
Exp 211	-1	1	-1	-1	1	-1	1	1	0.557
Exp 212	1	1	-1	-1	1	-1	1	1	0.538
Exp 213	-1	-1	1	-1	1	-1	1	1	0.557
Exp 214	1	-1	1	-1	1	-1	1	1	0.538
Exp 215	-1	1	1	-1	1	-1	1	1	0.558

Exp 216	1	1	1	-1	1	-1	1	1	0.538
Exp 217	-1	-1	-1	1	1	-1	1	1	0.538
Exp 218	1	-1	-1	1	1	-1	1	1	0.558
Exp 219	-1	1	-1	1	1	-1	1	1	0.538
Exp 220	1	1	-1	1	1	-1	1	1	0.557
Exp 221	-1	-1	1	1	1	-1	1	1	0.538
Exp 222	1	-1	1	1	1	-1	1	1	0.557
Exp 223	-1	1	1	1	1	-1	1	1	0.538
Exp 224	1	1	1	1	1	-1	1	1	0.555
Exp 225	-1	-1	-1	-1	-1	1	1	1	0.555
Exp 226	1	-1	-1	-1	-1	1	1	1	0.538
Exp 227	-1	1	-1	-1	-1	1	1	1	0.557
Exp 228	1	1	-1	-1	-1	1	1	1	0.538
Exp 229	-1	-1	1	-1	-1	1	1	1	0.557
Exp 230	1	-1	1	-1	-1	1	1	1	0.538
Exp 231	-1	1	1	-1	-1	1	1	1	0.558
Exp 232	1	1	1	-1	-1	1	1	1	0.538
Exp 233	-1	-1	-1	1	-1	1	1	1	0.538
Exp 234	1	-1	-1	1	-1	1	1	1	0.558
Exp 235	-1	1	-1	1	-1	1	1	1	0.538
Exp 236	1	1	-1	1	-1	1	1	1	0.557
Exp 237	-1	-1	1	1	-1	1	1	1	0.538
Exp 238	1	-1	1	1	-1	1	1	1	0.557
Exp 239	-1	1	1	1	-1	1	1	1	0.538
Exp 240	1	1	1	1	-1	1	1	1	0.555
Exp 241	-1	-1	-1	-1	1	1	1	1	0.557
Exp 242	1	-1	-1	-1	1	1	1	1	0.538
Exp 243	-1	1	-1	-1	1	1	1	1	0.558
Exp 244	1	1	-1	-1	1	1	1	1	0.538
Exp 245	-1	-1	1	-1	1	1	1	1	0.558
Exp 246	1	-1	1	-1	1	1	1	1	0.538
Exp 247	-1	1	1	-1	1	1	1	1	0.560
Exp 248	1	1	1	-1	1	1	1	1	0.538
Exp 249	-1	-1	-1	1	1	1	1	1	0.538
Exp 250	1	-1	-1	1	1	1	1	1	0.557
Exp 251	-1	1	-1	1	1	1	1	1	0.538
Exp 252	1	1	-1	1	1	1	1	1	0.555
Exp 253	-1	-1	1	1	1	1	1	1	0.538
Exp 254	1	-1	1	1	1	1	1	1	0.555
Exp 255	-1	1	1	1	1	1	1	1	0.538
Exp 256	1	1	1	1	1	1	1	1	0.554

Full factorial experiments

	W1	W2	W3	W4	W5	W6	W7	W8	**Total Error**
Exp 1	-1	-1	-1	-1	-1	-1	-1	-1	1.290
Exp 2	1	-1	-1	-1	-1	-1	-1	-1	1.449
Exp 3	-1	1	-1	-1	-1	-1	-1	-1	1.209
Exp 4	1	1	-1	-1	-1	-1	-1	-1	1.381
Exp 5	-1	-1	1	-1	-1	-1	-1	-1	1.381
Exp 6	1	-1	1	-1	-1	-1	-1	-1	1.530
Exp 7	-1	1	1	-1	-1	-1	-1	-1	1.300
Exp 8	1	1	1	-1	-1	-1	-1	-1	1.462
Exp 9	-1	-1	-1	1	-1	-1	-1	-1	1.449
Exp 10	1	-1	-1	1	-1	-1	-1	-1	1.584
Exp 11	-1	1	-1	1	-1	-1	-1	-1	1.381
Exp 12	1	1	-1	1	-1	-1	-1	-1	1.530
Exp 13	-1	-1	1	1	-1	-1	-1	-1	1.530
Exp 14	1	-1	1	1	-1	-1	-1	-1	1.653
Exp 15	-1	1	1	1	-1	-1	-1	-1	1.462
Exp 16	1	1	1	1	-1	-1	-1	-1	1.598
Exp 17	-1	-1	-1	-1	1	-1	-1	-1	1.209
Exp 18	1	-1	-1	-1	1	-1	-1	-1	1.381
Exp 19	-1	1	-1	-1	1	-1	-1	-1	1.119
Exp 20	1	1	-1	-1	1	-1	-1	-1	1.300
Exp 21	-1	-1	1	-1	1	-1	-1	-1	1.300
Exp 22	1	-1	1	-1	1	-1	-1	-1	1.462
Exp 23	-1	1	1	-1	1	-1	-1	-1	1.209
Exp 24	1	1	1	-1	1	-1	-1	-1	1.381
Exp 25	-1	-1	-1	1	1	-1	-1	-1	1.381
Exp 26	1	-1	-1	1	1	-1	-1	-1	1.530
Exp 27	-1	1	-1	1	1	-1	-1	-1	1.300
Exp 28	1	1	-1	1	1	-1	-1	-1	1.462
Exp 29	-1	-1	1	1	1	-1	-1	-1	1.462
Exp 30	1	-1	1	1	1	-1	-1	-1	1.598
Exp 31	-1	1	1	1	1	-1	-1	-1	1.381
Exp 32	1	1	1	1	1	-1	-1	-1	1.530
Exp 33	-1	-1	-1	-1	-1	1	-1	-1	1.381
Exp 34	1	-1	-1	-1	-1	1	-1	-1	1.530
Exp 35	-1	1	-1	-1	-1	1	-1	-1	1.300
Exp 36	1	1	-1	-1	-1	1	-1	-1	1.462
Exp 37	-1	-1	1	-1	-1	1	-1	-1	1.462
Exp 38	1	-1	1	-1	-1	1	-1	-1	1.598
Exp 39	-1	1	1	-1	-1	1	-1	-1	1.381
Exp 40	1	1	1	-1	-1	1	-1	-1	1.530
Exp 41	-1	-1	-1	1	-1	1	-1	-1	1.530
Exp 42	1	-1	-1	1	-1	1	-1	-1	1.653
Exp 43	-1	1	-1	1	-1	1	-1	-1	1.462
Exp 44	1	1	-1	1	-1	1	-1	-1	1.598
Exp 45	-1	-1	1	1	-1	1	-1	-1	1.598
Exp 46	1	-1	1	1	-1	1	-1	-1	1.707
Exp 47	-1	1	1	1	-1	1	-1	-1	1.530
Exp 48	1	1	1	1	-1	1	-1	-1	1.653
Exp 49	-1	-1	-1	-1	1	1	-1	-1	1.300

Exp 50	1	-1	-1	-1	1	1	-1	-1	1.462
Exp 51	-1	1	-1	-1	1	1	-1	-1	1.209
Exp 52	1	1	-1	-1	1	1	-1	-1	1.381
Exp 53	-1	-1	1	-1	1	1	-1	-1	1.381
Exp 54	1	-1	1	-1	1	1	-1	-1	1.530
Exp 55	-1	1	1	-1	1	1	-1	-1	1.290
Exp 56	1	1	1	-1	1	1	-1	-1	1.449
Exp 57	-1	-1	-1	1	1	1	-1	-1	1.462
Exp 58	1	-1	-1	1	1	1	-1	-1	1.598
Exp 59	-1	1	-1	1	1	1	-1	-1	1.381
Exp 60	1	1	-1	1	1	1	-1	-1	1.530
Exp 61	-1	-1	1	1	1	1	-1	-1	1.530
Exp 62	1	-1	1	1	1	1	-1	-1	1.653
Exp 63	-1	1	1	1	1	1	-1	-1	1.449
Exp 64	1	1	1	1	1	1	-1	-1	1.584
Exp 65	-1	-1	-1	-1	-1	-1	1	-1	1.000
Exp 66	1	-1	-1	-1	-1	-1	1	-1	0.812
Exp 67	-1	1	-1	-1	-1	-1	1	-1	1.094
Exp 68	1	1	-1	-1	-1	-1	1	-1	0.906
Exp 69	-1	-1	1	-1	-1	-1	1	-1	0.906
Exp 70	1	-1	1	-1	-1	-1	1	-1	0.724
Exp 71	-1	1	1	-1	-1	-1	1	-1	1.000
Exp 72	1	1	1	-1	-1	-1	1	-1	0.818
Exp 73	-1	-1	-1	1	-1	-1	1	-1	1.188
Exp 74	1	-1	-1	1	-1	-1	1	-1	1.000
Exp 75	-1	1	-1	1	-1	-1	1	-1	1.276
Exp 76	1	1	-1	1	-1	-1	1	-1	1.094
Exp 77	-1	-1	1	1	-1	-1	1	-1	1.094
Exp 78	1	-1	1	1	-1	-1	1	-1	0.906
Exp 79	-1	1	1	1	-1	-1	1	-1	1.182
Exp 80	1	1	1	1	-1	-1	1	-1	1.000
Exp 81	-1	-1	-1	-1	1	-1	1	-1	0.906
Exp 82	1	-1	-1	-1	1	-1	1	-1	0.724
Exp 83	-1	1	-1	-1	1	-1	1	-1	1.000
Exp 84	1	1	-1	-1	1	-1	1	-1	0.812
Exp 85	-1	-1	1	-1	1	-1	1	-1	0.812
Exp 86	1	-1	1	-1	1	-1	1	-1	0.637
Exp 87	-1	1	1	-1	1	-1	1	-1	0.906
Exp 88	1	1	1	-1	1	-1	1	-1	0.724
Exp 89	-1	-1	-1	1	1	-1	1	-1	1.094
Exp 90	1	-1	-1	1	1	-1	1	-1	0.906
Exp 91	-1	1	-1	1	1	-1	1	-1	1.188
Exp 92	1	1	-1	1	1	-1	1	-1	1.000
Exp 93	-1	-1	1	1	1	-1	1	-1	1.000
Exp 94	1	-1	1	1	1	-1	1	-1	0.812
Exp 95	-1	1	1	1	1	-1	1	-1	1.094
Exp 96	1	1	1	1	1	-1	1	-1	0.906
Exp 97	-1	-1	-1	-1	-1	1	1	-1	1.094
Exp 98	1	-1	-1	-1	-1	1	1	-1	0.906
Exp 99	-1	1	-1	-1	-1	1	1	-1	1.188
Exp 100	1	1	-1	-1	-1	1	1	-1	1.000
Exp 101	-1	-1	1	-1	-1	1	1	-1	1.000
Exp 102	1	-1	1	-1	-1	1	1	-1	0.812
Exp 103	-1	1	1	-1	-1	1	1	-1	1.094
Exp 104	1	1	1	-1	-1	1	1	-1	0.906

Exp 105	-1	-1	-1	1	-1	1	1	-1	1.276
Exp 106	1	-1	-1	1	-1	1	1	-1	1.094
Exp 107	-1	1	-1	1	-1	1	1	-1	1.363
Exp 108	1	1	-1	1	-1	1	1	-1	1.188
Exp 109	-1	-1	1	1	-1	1	1	-1	1.188
Exp 110	1	-1	1	1	-1	1	1	-1	1.000
Exp 111	-1	1	1	1	-1	1	1	-1	1.276
Exp 112	1	1	1	1	-1	1	1	-1	1.094
Exp 113	-1	-1	-1	-1	1	1	1	-1	1.000
Exp 114	1	-1	-1	-1	1	1	1	-1	0.818
Exp 115	-1	1	-1	-1	1	1	1	-1	1.094
Exp 116	1	1	-1	-1	1	1	1	-1	0.906
Exp 117	-1	-1	1	-1	1	1	1	-1	0.906
Exp 118	1	-1	1	-1	1	1	1	-1	0.724
Exp 119	-1	1	1	-1	1	1	1	-1	1.000
Exp 120	1	1	1	-1	1	1	1	-1	0.812
Exp 121	-1	-1	-1	1	1	1	1	-1	1.182
Exp 122	1	-1	-1	1	1	1	1	-1	1.000
Exp 123	-1	1	-1	1	1	1	1	-1	1.276
Exp 124	1	1	-1	1	1	1	1	-1	1.094
Exp 125	-1	-1	1	1	1	1	1	-1	1.094
Exp 126	1	-1	1	1	1	1	1	-1	0.906
Exp 127	-1	1	1	1	1	1	1	-1	1.188
Exp 128	1	1	1	1	1	1	1	-1	1.000
Exp 129	-1	-1	-1	-1	-1	-1	-1	1	1.000
Exp 130	1	-1	-1	-1	-1	-1	-1	1	1.188
Exp 131	-1	1	-1	-1	-1	-1	-1	1	0.906
Exp 132	1	1	-1	-1	-1	-1	-1	1	1.094
Exp 133	-1	-1	1	-1	-1	-1	-1	1	1.094
Exp 134	1	-1	1	-1	-1	-1	-1	1	1.276
Exp 135	-1	1	1	-1	-1	-1	-1	1	1.000
Exp 136	1	1	1	-1	-1	-1	-1	1	1.182
Exp 137	-1	-1	-1	1	-1	-1	-1	1	0.812
Exp 138	1	-1	-1	1	-1	-1	-1	1	1.000
Exp 139	-1	1	-1	1	-1	-1	-1	1	0.724
Exp 140	1	1	-1	1	-1	-1	-1	1	0.906
Exp 141	-1	-1	1	1	-1	-1	-1	1	0.906
Exp 142	1	-1	1	1	-1	-1	-1	1	1.094
Exp 143	-1	1	1	1	-1	-1	-1	1	0.818
Exp 144	1	1	1	1	-1	-1	-1	1	1.000
Exp 145	-1	-1	-1	-1	1	-1	-1	1	1.094
Exp 146	1	-1	-1	-1	1	-1	-1	1	1.276
Exp 147	-1	1	-1	-1	1	-1	-1	1	1.000
Exp 148	1	1	-1	-1	1	-1	-1	1	1.188
Exp 149	-1	-1	1	-1	1	-1	-1	1	1.188
Exp 150	1	-1	1	-1	1	-1	-1	1	1.363
Exp 151	-1	1	1	-1	1	-1	-1	1	1.094
Exp 152	1	1	1	-1	1	-1	-1	1	1.276
Exp 153	-1	-1	-1	1	1	-1	-1	1	0.906
Exp 154	1	-1	-1	1	1	-1	-1	1	1.094
Exp 155	-1	1	-1	1	1	-1	-1	1	0.812
Exp 156	1	1	-1	1	1	-1	-1	1	1.000
Exp 157	-1	-1	1	1	1	-1	-1	1	1.000
Exp 158	1	-1	1	1	1	-1	-1	1	1.188
Exp 159	-1	1	1	1	1	-1	-1	1	0.906

Exp 160	1	1	1	1	1	-1	-1	1	1.094
Exp 161	-1	-1	-1	-1	-1	1	-1	1	0.906
Exp 162	1	-1	-1	-1	-1	1	-1	1	1.094
Exp 163	-1	1	-1	-1	-1	1	-1	1	0.812
Exp 164	1	1	-1	-1	-1	1	-1	1	1.000
Exp 165	-1	-1	1	-1	-1	1	-1	1	1.000
Exp 166	1	-1	1	-1	-1	1	-1	1	1.188
Exp 167	-1	1	1	-1	-1	1	-1	1	0.906
Exp 168	1	1	1	-1	-1	1	-1	1	1.094
Exp 169	-1	-1	-1	1	-1	1	-1	1	0.724
Exp 170	1	-1	-1	1	-1	1	-1	1	0.906
Exp 171	-1	1	-1	1	-1	1	-1	1	0.637
Exp 172	1	1	-1	1	-1	1	-1	1	0.812
Exp 173	-1	-1	1	1	-1	1	-1	1	0.812
Exp 174	1	-1	1	1	-1	1	-1	1	1.000
Exp 175	-1	1	1	1	-1	1	-1	1	0.724
Exp 176	1	1	1	1	-1	1	-1	1	0.906
Exp 177	-1	-1	-1	-1	1	1	-1	1	1.000
Exp 178	1	-1	-1	-1	1	1	-1	1	1.182
Exp 179	-1	1	-1	-1	1	1	-1	1	0.906
Exp 180	1	1	-1	-1	1	1	-1	1	1.094
Exp 181	-1	-1	1	-1	1	1	-1	1	1.094
Exp 182	1	-1	1	-1	1	1	-1	1	1.276
Exp 183	-1	1	1	-1	1	1	-1	1	1.000
Exp 184	1	1	1	-1	1	1	-1	1	1.188
Exp 185	-1	-1	-1	1	1	1	-1	1	0.818
Exp 186	1	-1	-1	1	1	1	-1	1	1.000
Exp 187	-1	1	-1	1	1	1	-1	1	0.724
Exp 188	1	1	-1	1	1	1	-1	1	0.906
Exp 189	-1	-1	1	1	1	1	-1	1	0.906
Exp 190	1	-1	1	1	1	1	-1	1	1.094
Exp 191	-1	1	1	1	1	1	-1	1	0.812
Exp 192	1	1	1	1	1	1	-1	1	1.000
Exp 193	-1	-1	-1	-1	-1	-1	1	1	0.710
Exp 194	1	-1	-1	-1	-1	-1	1	1	0.551
Exp 195	-1	1	-1	-1	-1	-1	1	1	0.791
Exp 196	1	1	-1	-1	-1	-1	1	1	0.619
Exp 197	-1	-1	1	-1	-1	-1	1	1	0.619
Exp 198	1	-1	1	-1	-1	-1	1	1	0.470
Exp 199	-1	1	1	-1	-1	-1	1	1	0.700
Exp 200	1	1	1	-1	-1	-1	1	1	0.538
Exp 201	-1	-1	-1	1	-1	-1	1	1	0.551
Exp 202	1	-1	-1	1	-1	-1	1	1	0.416
Exp 203	-1	1	-1	1	-1	-1	1	1	0.619
Exp 204	1	1	-1	1	-1	-1	1	1	0.470
Exp 205	-1	-1	1	1	-1	-1	1	1	0.470
Exp 206	1	-1	1	1	-1	-1	1	1	0.347
Exp 207	-1	1	1	1	-1	-1	1	1	0.538
Exp 208	1	1	1	1	-1	-1	1	1	0.402
Exp 209	-1	-1	-1	-1	1	-1	1	1	0.791
Exp 210	1	-1	-1	-1	1	-1	1	1	0.619
Exp 211	-1	1	-1	-1	1	-1	1	1	0.881
Exp 212	1	1	-1	-1	1	-1	1	1	0.700
Exp 213	-1	-1	1	-1	1	-1	1	1	0.700
Exp 214	1	-1	1	-1	1	-1	1	1	0.538

Exp 215	-1	1	1	-1	1	-1	1	1	0.791
Exp 216	1	1	1	-1	1	-1	1	1	0.619
Exp 217	-1	-1	-1	1	1	-1	1	1	0.619
Exp 218	1	-1	-1	1	1	-1	1	1	0.470
Exp 219	-1	1	-1	1	1	-1	1	1	0.700
Exp 220	1	1	-1	1	1	-1	1	1	0.538
Exp 221	-1	-1	1	1	1	-1	1	1	0.538
Exp 222	1	-1	1	1	1	-1	1	1	0.402
Exp 223	-1	1	1	1	1	-1	1	1	0.619
Exp 224	1	1	1	1	1	-1	1	1	0.470
Exp 225	-1	-1	-1	-1	-1	1	1	1	0.619
Exp 226	1	-1	-1	-1	-1	1	1	1	0.470
Exp 227	-1	1	-1	-1	-1	1	1	1	0.700
Exp 228	1	1	-1	-1	-1	1	1	1	0.538
Exp 229	-1	-1	1	-1	-1	1	1	1	0.538
Exp 230	1	-1	1	-1	-1	1	1	1	0.402
Exp 231	-1	1	1	-1	-1	1	1	1	0.619
Exp 232	1	1	1	-1	-1	1	1	1	0.470
Exp 233	-1	-1	-1	1	-1	1	1	1	0.470
Exp 234	1	-1	-1	1	-1	1	1	1	0.347
Exp 235	-1	1	-1	1	-1	1	1	1	0.538
Exp 236	1	1	-1	1	-1	1	1	1	0.402
Exp 237	-1	-1	1	1	-1	1	1	1	0.402
Exp 238	1	-1	1	1	-1	1	1	1	**0.293**
Exp 239	-1	1	1	1	-1	1	1	1	0.470
Exp 240	1	1	1	1	-1	1	1	1	0.347
Exp 241	-1	-1	-1	-1	1	1	1	1	0.700
Exp 242	1	-1	-1	-1	1	1	1	1	0.538
Exp 243	-1	1	-1	-1	1	1	1	1	0.791
Exp 244	1	1	-1	-1	1	1	1	1	0.619
Exp 245	-1	-1	1	-1	1	1	1	1	0.619
Exp 246	1	-1	1	-1	1	1	1	1	0.470
Exp 247	-1	1	1	-1	1	1	1	1	0.710
Exp 248	1	1	1	-1	1	1	1	1	0.551
Exp 249	-1	-1	-1	1	1	1	1	1	0.538
Exp 250	1	-1	-1	1	1	1	1	1	0.402
Exp 251	-1	1	-1	1	1	1	1	1	0.619
Exp 252	1	1	-1	1	1	1	1	1	0.470
Exp 253	-1	-1	1	1	1	1	1	1	0.470
Exp 254	1	-1	1	1	1	1	1	1	0.347
Exp 255	-1	1	1	1	1	1	1	1	0.551
Exp 256	1	1	1	1	1	1	1	1	0.416