Engineering Reliable Neural Network Systems

by

Christopher Hugh Messom

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the degree of

Doctor of Philosophy in Computer Studies

of the Loughborough University of Technology

November 1992.

Name: Christopher Hugh Messom

Definitive Title:-

Engineering Reliable Neural Network Systems

Synopsis:-

This thesis presents a study of neural network representation and behaviour. The study places neural networks in the context of designing reliable systems. Several new results on network size and topology are presented.

Knowledge based training of neural networks is examined. This is essential for designing reliable neural systems in which the subsymbolic reasoning processes are well defined. Sandwich nodes are introduced and studied as atomic knowledge elements in neural networks. Two new network architectures are introduced, the Loughborough Net and the Loughborough Control Net. These make use of the parallelism inherent in sandwich node representations.

The interpretation of neural network representations as logical transformations and rule systems are presented. An equivalence of the rule systems and neural network representation is proposed and discussed. This equivalence is required in order that the total behaviour of the neural network can be understood.

A new methodology for designing reliable neural network systems making use of knowledge based training is proposed. This is used to present a general design methodology for the construction of reliable neural network control systems using the Loughborough Control Net architecture. A case study is discussed where the methodology was applied to the design of an adhesive dispensing controller.

Acknowledgements

I would like to thank;

Dr Christopher Hinde, my supervisor for his help and support throughout my study.

Professor E.A. Edmonds, my director of research.

Dr A.A. West and Professor D. Williams of the Manufacturing Engineering Department of the Loughborough University of Technology for collaborating on the implementation of a neural network controller for an adhesive dispensing process in the manufacture of mixed technology printed circuit boards.

Contents

Appendices

List of Figures

List of Tables

List of Graphs

List of Photographs

Part I. Introductory Ideas

Chapter 1. Introduction and Background

Human history has been a record of the design and appliance of tools, from the bone clubs and flint blades of prehistoric man to the computerised automated systems of modern man (Bronowski '73, Birdsall & Cipolla '79). The tools that were available extended the power and experience of man and in turn motivated the development of ever more sophisticated technology (Forbes & Dijksterius '63).

The tools that have stimulated the development of the mind have led to the peaks of human achievement, unmanned space exploration being just one example (Clark '85, Lilley '65). As the tasks that man attempted became greater and more ambitious, calculating systems and machines were developed. Symbolic mathematics provided a framework for abstract reasoning while the use of predicate logic formalised linguistic reasoning and argument.

The development of the electronic computer revolutionised calculating and reasoning systems. Automating logical reasoning allowed evermore complex systems to be developed, leading to the design and construction of automated machines.

Truly automated processes would require no human operator or controller. This may seem attractive, especially for very mundane activities. However any process, automated or otherwise must be sufficiently understood to ensure the safety and reliability of the system. A human controller or agent has knowledge about the behaviour of the automated process and can therefore predict the behaviour of the system under various conditions. This human agent can then ensure the reliability of the system. This is especially desirable in the case of safety critical applications (Warwick & Tham '91).

Outline of chapter

This chapter provides a historical background and systematic development of the basic ideas of the field of neural network systems. The place of formal logic in automated systems is discussed. Historically it has been the motivating force behind the pursuit of automated systems and provides a foundation on which more complex systems can be built.

Neural networks are introduced as a possible solution to the performance limitations of logical systems. The general monotonic and incremental nature of logical systems effectively places a bound on performance in real time. As the knowledge base becomes ever larger, the response time of the system will increase. Neural network techniques offer a solution to this problem via the inherent parallel nature of neural network execution. However large the network becomes, the response time of a network implemented in hardware is only dependent on the depth (the number of layers) of the feedforward network and not the number of nodes in the network. Difficulties exist with neural network representation and behaviour, which provides the motivation for this thesis. The motivating issues of this thesis are discussed in chapter two.

The basic properties of neural network systems are introduced in this chapter, highlighting their representational limitations. A feedforward network does not have the representational power of a first order logical system, while a recurrent network has the same representational capability as the first order logical system. These questions are relevant for the design considerations of neural systems.

The various different network architectures, recurrent and feedforward networks and the training techniques they employ are introduced. They are discussed in further depth in chapter three and four.

Overview of reasoning with logic

Reasoning systems have existed for many centuries, since the time of the Greeks with Aristotelian reasoning to the work of Boole('47 & '58), Carroll('58 & '77), Keynes('06) and Russell('03 & '19) in more recent history. These symbolic systems provided an excellent framework for reasoning through complex arguments, however most failed when confronted with uncertainty.

Everyday linguistic reasoning was seen not to obey many of the properties of these formal systems, namely that of consistency and the conditions for ambiguity. Similarly problem solving and spatial reasoning could not be modelled by these systems, limiting their overall usefulness.

The work of Keynes('21) and Venn('94) on probability provided a foundation for working in systems where certainty was not guaranteed and unknown elements existed. Automating these problems proved difficult and significant extensions and variations were required to overcome them.

Many of the issues addressed in this thesis have existed for many years. Early civilisations were interested in thought processes and the mechanisms of reasoning. Traditional or Aristotelian logic was developed in the time of the Greeks (Boyer & Merzbach '89) and is thought to have been used in the training of the lawyers and magistrates and by extension the politicians of the city states.

Significant traditions of logic in Arab, Indian and Chinese civilisations also existed (Mikami '74). These traditions spawned algebraic logic which separated logic from its use and misuse in language and argument. Boole('47) with "The mathematical analysis of logic" and De Morgan('47) with "Formal logic" provided a wider structure in which logic could be studied.

The formal logic as studied, provided a strong mathematical foundation for symbolic reasoning, which was amenable to automation. One of the limitations that resulted was its abstraction from everyday reasoning and the inherent uncertainties that result. Only dichotic, true- false systems were considered, partly due to its simplicity of representation and execution.

Logical systems

Logic is the oldest formal system for reasoning, which allows natural language like statements to be manipulated so proving whether they are consistent or not. Proofs of queries could be given by showing that the negation of the query and the statements in the database were inconsistent (reductio ad absurdum).

These logical systems were so successful in providing reasoning systems, that it formed the basis for many computerised reasoning machines. One of the most successful using only statements in clausal form is Prolog (Clocksin et al'87 and Shapiro et al'86) which allows queries to be made on a database. A depth first search strategy is used and a relevance condition is required on the search, so allowing the system to behave correctly even if an inconsistency exists in the database of clauses.

First order propositional logic is a system in which formulae exist employing atoms which take the values either true or false, with the logical connectives and, or, not and implication, as well as more intricate operators made from a combination of these. The calculus of the logical operations provide the ability to give absolute truth values to the formulae.

Reasoning processes can occur due to the implication symbol, defined in table 1.1.

		B	
	A → B	T	F
A	T	T	F
	F	T	T

Table 1.1 Truth table for the implication operator

With modus ponens (fig 1.1a)we have the reasoning process reaching the conclusions shown. With modus tollens (fig 1.1b) we have a variation.

$$\begin{array}{c} a \\ b \leftarrow a \\ \hline \text{then} \quad b \end{array} \qquad\qquad \begin{array}{c} b \leftarrow a \\ \text{not}(b) \\ \hline \text{then} \quad \text{not}(a) \end{array}$$

Fig 1.1 a. Modus ponens reasoning, b. Modus tollens reasoning

The logical statements form a knowledge base that may be accessed. To find out if a given statement, the query, is consistent with the set of logical statements, the knowledge base, it is necessary to construct a proof of the statement.

This can be a particularly complex procedure, considering the variety of consistent transformations that can be made on the knowledge base. The procedure of "reductio ad absurdum" overcomes these problems. This procedure consists of adding the negation of the query to the knowledge base and then search for a contradiction. If a contradiction arises, that is the existence of both a true and a false instantiation of the same formula, then it can be assumed that it is due to the addition of the query formula. That is the negation of the query gives rise to the inconsistency and so the query must be consistent with the knowledge base. The query is proved.

Problems that could arise include the possibility that the set of logical statements are already inconsistent, that is contains a contradiction. The previous proof procedure would prove anything true with this knowledge base, even (a and not(a)). To overcome this difficulty, several methods can be employed. Maintaining the integrity of the knowledge base is one, but for large real world systems this may be particularly difficult. The second method is to maintain a relevancy criterion to the proof procedure. The query depends on several atoms and if the contradiction in the knowledge base is discovered to depend on a completely disjoint set of atoms then the inconsistency can be seen to be irrelevant to the query. This second method means that only relevant formulae are used in the proof procedure, reducing its complexity greatly.

Clausal form

However much the relevancy criterion reduces the complexity of proof procedures in first order logical systems, the variety of manipulations and transformations that can be applied to the formulae ensure that the proof procedure is in general difficult. Methods to overcome this problem centre around using a reduced subset of first order logic, such that representative power is still maintained while the proof procedures are simplified.

Horn clauses are such a reduced set of logic. Their structure is that of an implication, the head of the clause being implied by the antecedents. Kowalski('80) provides a comprehensive study of such systems. The proof procedure reduces to that of graph search and with suitable search algorithms ensures that the proofs can be completed.

Kowalski demonstrates how the clausal structure can be exploited in a logical problem solving environment. The approach is declarative, putting the solution of problems in logical bounds, leaving the search algorithms to provide the proofs to the solutions.

Benefits of logical systems

The greatest advantage of the propositional logic systems are the incremental nature of the knowledge structure. Large problems can be structured using subcomponents which are modelled separately and then recombined. The knowledge is added incrementally and monotonically. Even with systems of nonmonotonic reasoning great restructuring of the knowledge base is not necessary. As time goes by the knowledge based system improves incrementally and equally increases incrementally.

As the size of the knowledge base increases and the reasoning processes become more complex (as in uncertain reasoning), this incremental nature of the knowledge base approach becomes a liability. The size and computing power required to model the ever increasing body of knowledge has meant that something radical must be done. Increasing the performance of single processors and memory chips is not the solution since they have almost reached their physical limits (Hwang & Briggs '85). As the cost of processors decrease the competitive advantage will be to those automated systems that efficiently employ many processing units. This means that approaches that make use of multiple cooperating processors will be of greater advantage.

Making use of only a few processors would require complex processors with sophisticated communication to solve significant problems (as in the distributed computing approach with its message passing environment (Sloman & Kramer '87)). Making use of many simple processors on the other hand will admit simple connection and communication.

This approach is pursued by neural network systems.

The processing elements in neural systems are all identical and extremely simple in operation. The connection pattern is dense, each element being connected to many others. The message passing is simple consisting of a scalar function of activity. Although each processing unit is simple the behaviour of the total system can be quite complex. With the existence of automated training algorithms, system implementation can successfully be realised.

Connectionist models and neural networks

The main motivation for studying distributed representations of knowledge is the experimental field of the neurosciences (Amari '77 and Amari et al '77). Work in the neurosciences showed that the brain was constructed via the interconnection of neurons (Kolb et al '80). Neurons being on the face of it, particularly simple units. Identifying the functions of individual neurons proved impossible, the specific function being distributed over a pattern of activity of many neurons.

The human mind can perform feats that modern computers with sophisticated programming can not as yet match. The hardware available to the programmer however seems far superior, being about 10^6 times as fast at performing a single computation than the human mind (Kolb et al '80). The human mind makes up for this speed disadvantage via a highly parallel mechanism of execution. Evidence suggests this parallelism is implemented in a distributed representation, namely that the various concepts are stored as a pattern of activity over a distributed region of the brain.

Armed with these motivating elements, a working model, that is the brain, and a means of constructing new models, on sophisticated digital computers, researchers have invested considerable time and effort in achieving some notable results. These will be outlined further, starting from the original work on perceptrons through simple neural nets to sophisticated representational schemes making use of Bolzmann machines.

Success in knowledge representations via production rules and logical systems has so far outreached that of connectionist models. However the neural techniques are improving, ultimately trying to reach the stage where significant problems can be solved in the 100 time step limit, that achieved by the brain, as proposed by Feldman('82).

Distributed representation

A local representation is one in which discrete elements are used to represent unique discrete functions. Within the connectionist framework this generally refers to the custom built neural networks where each node is considered to represent some particular function. This is seen in the Willshaw's ('81) grapheme/ word set semantic sememe feed forward net and McClelland's ('85) word perception model.

Distributed representations are very different. The network topology of the two nets, one with local the other with distributed representation, may look very similar, however they will essentially be distinct. A distributed representation, represents functions as a pattern of activity over a variety of units which themselves may be involved in representing distinct functions under a different pattern of activity (Feldman et al '82).

The difficulty of the distributed representation is in comprehending the meaning and behaviour of any particular net. Static observation of the nodes and weights would yield little information since the concepts are at a higher level, and since they interact in a distributed manner it would be almost impossible to decipher any meaningful structures. Observing the net at work will yield some of the meaning hidden in the units, but may still defy analysis of the complex interacting concepts driving the net.

There are many advantages in a distributed representation, robustness, general learning mechanisms and graceful degradation. However the problem of understanding the distributed representation must be resolved. This thesis formalises simple distributed representations so providing a mechanism for interpreting their function and behaviour.

Structure of neural models

The principle behind connectionist or neural models is that given particular inputs the model, the neural net, propagates a pattern of activity across its structure producing an output pattern. There is no attempt at explicit manipulation of the input as in conventional computational techniques.

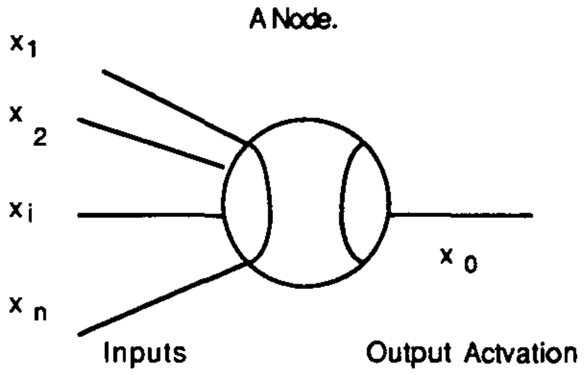


Fig 1.2 a. Single node with inputs and linearly summed output

This is facilitated by the netlike structure of connectionist models, constructed from simple node units and connection paths. The node units can be of various types, but the most studied have been of linear summation type. I.e the output activation is the sum of the input values. $X_0 = \Sigma^n{}_{i=1} X_i$. And variations have included linear summation threshold units or linear threshold squashing units, respectively $X_0 = \Sigma^n{}_{i=1} X_i - B$ (ß is the threshold bias for the node) and $X_0 = f_0(\Sigma^n{}_{i=1} X_i)$ (f_0 is the squashing function). See fig 1.2a for a schematic representation of a node unit.

An added sophistication is the weights that are given to the connections between the various nodes. These provide an easy mechanism for changing the structure and behaviour of the net. Namely altering these weights, reducing the weight to zero being equivalent to removing a connection. Therefore the final input from one node to another node is equal to the output from the first times the weight governing that connection. See fig 1.2b for a schematic representation of a weighted node unit.

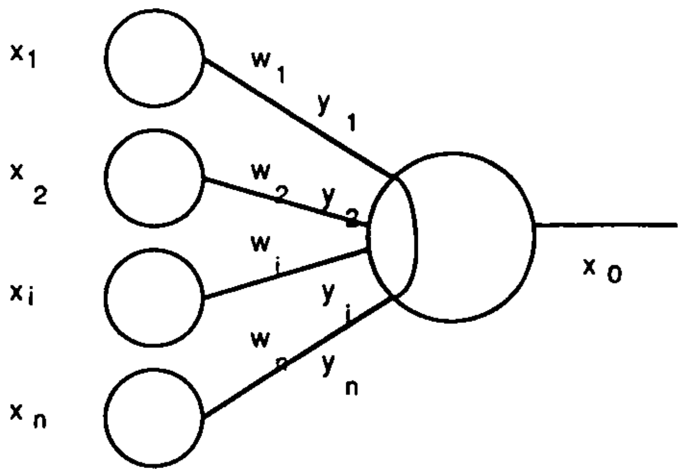


Fig 1.2 b. Single node with weighted inputs and thresholded output

Variable x_i is the output from the nodes, y_i is the input to the node, and w_i is the weight governing the connection.

Input to node on line i is $y_i = w_i * x_i$.

So for a general linear summation squashing unit $X_0 = f_0(\Sigma^n_{i=1} w_i * X_i)$.

Recurrent networks

Recurrent networks are fully interconnected neural networks. Fig 1.3 shows a simple recurrent network structure. There are no explicit input and outputs nodes. Each node associates a weight value with those to which it is connected. Hopfield formalised the associative net by considering the nodes as bipolar threshold units in which the outputs to each unit i, is fed back to each unit j, with the associated weight w_{ij} (Hopfield '84).

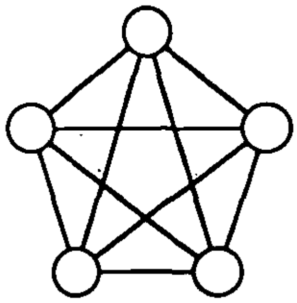


Fig 1.3 A recurrent network structure

To ensure the convergence of the net Hopfield used an iterative update procedure, with the added constraint that $w_{ij} = w_{ji}$. The capacity of the network must not be exceeded for this convergence criterion to be valid. The formal definition of the net was carried out as follows;

The weights were assigned as,

$$w_{ij} = \{ \Sigma^{N}_{y=0} x^{y}_{i} . x^{y}_{j} \quad i \neq j,$$
$$0 \quad i = j. \}$$

w_{ij} is the weight from node i to j, x^{y}_{i} is the i^{th} element of the training set y.

Therefore the behaviour of the net could be likened to minimising the energy function, $E = -\Sigma_{i<j} x^{y}_{i} . x^{y}_{j} . w_{ij} + \Sigma_{j} x_{j} . ø_{j}$. This follows since the derivative of this function represents the change in output at each stage of the update procedure that is employed in the convergence of the Hopfield net.

Essentially the structure of a Hopfield net is very simple, but its representational and behavioural power is fairly great. Its main function is that of an associative memory device or that of an optimiser. Its behaviour as an associative memory device is obvious, since it is a mathematical model of such devices (hard wired or theoretical), as discussed above. Given partial or noise affected input, the Hopfield net can extract the best example from the set of representative patterns, that it most closely matches.

The power of such devices is not limitless. Hopfield places a bound on the capacity of 0.15* the number of nodes as the maximum number of representative classes that can be stored, without degrading the performance to an unacceptable level.

Hopfield nets can also be applied to optimisation problems. Tank & Hopfield('86 & '86) demonstrate how a Hopfield net can be used in solving an optimisation problem in shelving rates and also a net solution to the travelling salesman problem. Both these problems require astute coding, namely the specification of inhibitory connections to limit impossible solutions. In the application of these nets to general problem solving, the question of convergence rates and efficiency in execution arise. Providing variations in the update schemes, would provide the possibility of improvement.

Another learning technique which stems from the biological sciences, is Hebbian learning. This involves rewarding connections to nodes that contribute most in activity. This means that nodes which contribute the greatest to the representative scheme receive the greater biases. Jacyna & Malaret('89) study the performance of a Hopfield net, using a Hebbian learning rule.

Associative memory devices represent a class of systems which require an input to be given which will on convergence remain stable. That is the input units are also the output units and these values (input and output), in the stable case, will be equal. Therefore, in such systems no complications arise due to considerations of non linear transformations since in one iterative cycle the transformation maintains the input.

In coding general functional transformations, the Hopfield net is of little use. Here we are considering the problem of receiving a relatively clean input vector and require an output vector that is a general Boolean transformation of the input. These transformations being non linear can not be coded in a network without hidden units. This can be seen when we examine the stable state criterion, see chapter four.

Bolzmann machines can be viewed as an extension to the Hopfield net scheme. The update is stochastic and there are hidden units which allow for general functional representation. The units in a Bolzmann machine take output values of 1 or 0. That is, they are active or inactive. The units take these values in proportion to its energy contribution, which is the sigmoid of the sum of the input values to that node.

That is, the probability that a unit is active is given by the formula;

$$\text{prob\_active(unit x)} = 1/(1 + \exp(-\Delta E_x/T),$$

where T is the temperature function, while ΔE_x is the energy contribution of the node x. This value is the sum of the inputs.

$$\Delta E_x = \Sigma_i \, w_{i\,x} . O_i \, ,$$

$w_{i\,x}$ is the weight from the i^{th} unit to unit x while O_i is the state of the i^{th} unit.

Given repeated application of the update rules the network reaches an equilibrium. This is termed the thermodynamic equilibrium. At high temperatures the network converges to equilibrium quickly, while at low temperatures this is not the case. High temperatures allow many high energy states, while low temperatures allow lower energy states to be more probable. Simulated annealing has been studied as a possible scheme to increase convergence while promoting low energy states.

The mathematics of the training scheme for Bolzmann machines proceeds in many ways similar to that of a gradient descent scheme. The weights are adjusted via the formula,

$$\Delta w_{i\,j} = \mu . (< O_i O_j >^+ - < O_i O_j >^-) \, ,$$

where $< O_i O_j >$ represents the probability that over all cases that unit i and j are both active, while the superscripts + and - represent the cases when the network has the output units clamped to their required value and are unclamped, respectively. Varying the learning rate ensures the global properties of the learning scheme are maintained as required. Small μ ensures that the scheme exhibits gradient descent.

Kirkpatrick et al('83) discuss the behaviour of statistical computational machines. They show that the implementation of Bolzmann machines and similar computational schemes are possible in hardware. They discuss an optimisation problem, the travelling salesman problem, to demonstrate some of the properties of the system.

Perceptrons

Perceptrons, particularly single layered threshold units have been extensively studied, see Minsky & Papert ('69).Their applications and particularly their limitations were well documented.

Perceptrons are feed forward nets with in general linear threshold units with a hard limiting squashing function (fig 1.4a). Single layer perceptrons have been shown to be able to classify objects into two regions separated by a hyperplane. Rosenblatt ('62) developed the perceptron convergence procedure which demonstrated this.

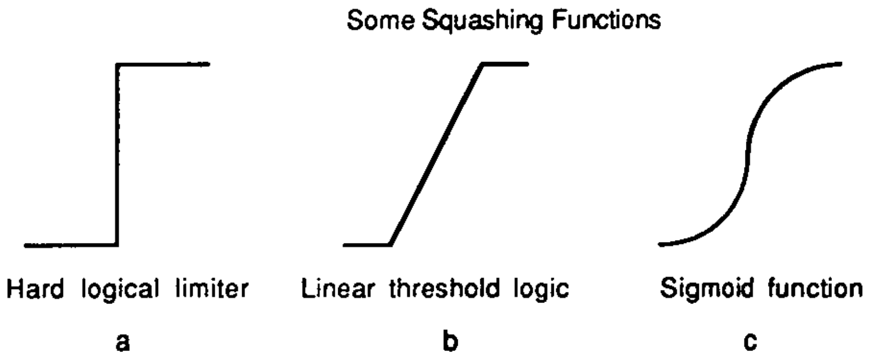


Fig 1.4 Squashing functions

Multilayer perceptrons lacked a learning procedure (Block '70) such as the one developed for single layer perceptrons and so were not extensively studied until recently. The back propagation algorithm provided a general learning scheme for the feed forward multilayer perceptrons using a sigmoid squashing function as seen in fig 1.4c (this is further discussed in chapter four). Although convergence for this and other schemes could not be guaranteed, many interesting properties could be observed.

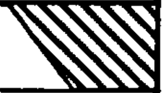
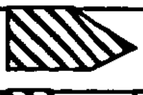
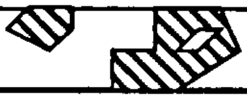
Structure	Type of decision region	Most general shape
Single layer	Half plane bounded by hyper plane	
Two layer	Convex regions	
Three layer	Arbitrary	

Table 1.2 Properties of perceptrons

The multilayer perceptrons use non linear nodes to gain full advantage of their structure, since a linear multilayer net can not do more than a single layer perceptron, (shown by the equivalence of a linear multilayer perceptron to a particular single layered one). Using a hard limiting squashing function (fig 1.4a), which is non linear, a two layer perceptron can solve the exclusive or problem (fig 1.5) and with multilayered perceptrons with suitable numbers of units, any decision region may be modelled (Table 1.2).

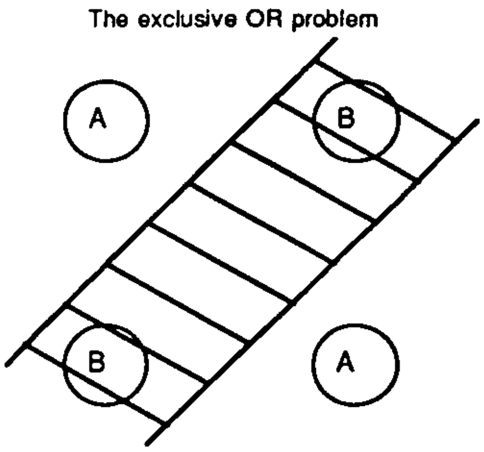


Fig 1.5 Segmentation of the input space to solve the "exclusive or" problem

Using the sigmoid squashing function can produce smoothly bound decision regions and so even the most general situations can be modelled using a reasonable number of units.

Feedforward networks

Feedforward neural networks have been developed from perceptron based approaches to multilayered systems.

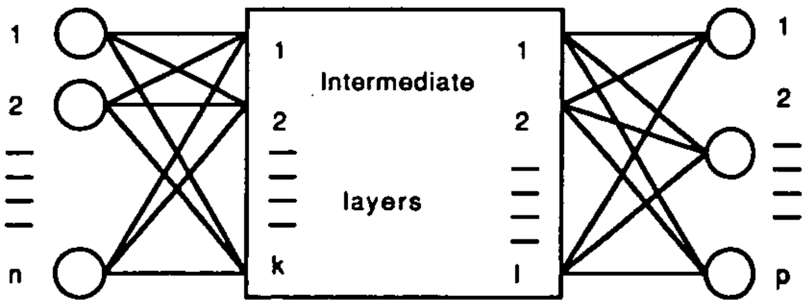


Fig 1.6 A feedforward network structure

A feed forward net refers to a net where the output from a unit at a particular level (one layer) goes to the input of nodes only at a subsequent level. That is the output can not go to any node in its own layer or a lower one. Therefore there are no loops or cyclic paths in the activation propagation. Fig 1.6 shows a feedforward network structure.

Given a feed forward network the back propagation algorithm may be used to produce a network after convergence with various interesting properties. Hinton ('86) demonstrated the power of this convergence scheme to discover underlying semantic features in a knowledge domain. Back propagation has also been used in nets for text to speech mapping (Sejnowski & Rosenberg '87) and phoneme recognition models. The power of feedforward net comes from their ability to represent general transformations, and the existence of training algorithms to train the net with a representative set of the transformation.

Feedforward neural networks are trained via the back propagation training scheme. A network is set up with weights that are small random numbers. The training set is applied to the net and the output values noted. The energy function for this procedure is the difference, squared and summed, between the required outputs and the actual outputs computed by the network. (Refer to the annotations of fig 1.7).

$E= 0.5^{*} \Sigma_{s,i}(aO_{s,i} - cO_{s,i})^2$, where $aO_{s,i}$ is the required output over output i given the training sample s.

Minimising this energy function would provide the best solution to the problem.

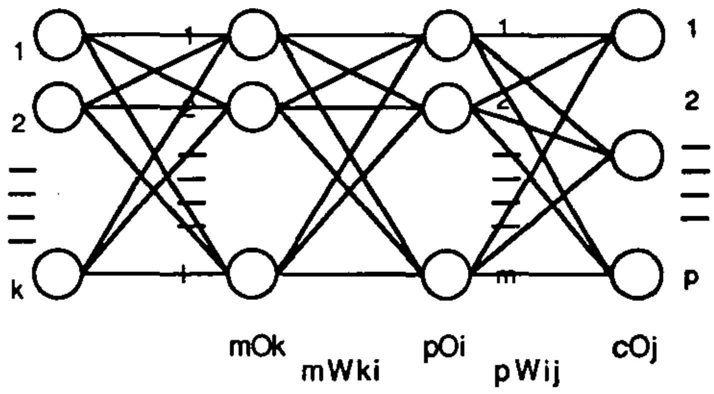


Fig 1.7 A multilayered feedforward network with annotated nodes and weights

The calculated outputs being a function of the weights, the problem of minimising the energy becomes that of varying the weights to solve the problem. Seeing the contribution of a particular weight change to the total energy provides a scheme of altering the weights, via gradient descent.

$\Delta w_{ij} = -\mu \; \partial E/\partial w_{ij}$.

It can be seen that this scheme for a small learning parameter μ will execute iterative hill climb. That is the scheme will find the nearest local optimum. Problems will obviously arise if this is not the global optimum and so several techniques exist to ensure that the search does not get stranded in unfavourable optima, while still retaining reasonably rapid convergence. The details of the back propagation algorithm are discussed in chapter four.

Vogl et al('88) discuss methods for ensuring rapid convergence of neural net. Parallel presentation of the training set is suggested as one method for ensuring true hill climbing properties of the training algorithm. Another method presented is that of varying the convergence rates depending on the performance of the algorithm. This involves manipulating the learning parameter in the particular manner required.

The performance of the algorithm is taken into account by increasing the parameter if the last update decreased the error function. A momentum term is also applied, (namely a factor proportional to the last update term, to ensure more rapid movement in the direction of convergence). If a step produces an increase in the error then the learning factor is decreased and the momentum term removed, (the last update did not provide an improvement).

$\Delta w_{ij}(m+1) = \mu \; \Sigma_p \partial E/\partial \; w_{ij} + \beta \; \Delta w_{ij}(m)$,

where μ is the learning rate, β the momentum term and m the iterative index.

Neural representations

From an external viewpoint the functional characteristics of neural network systems consist of the inputs that are applied to the system and the outputs that are received.

However, no knowledge of the internal structure and behaviour would exist. As the extensive literature in the field of expert systems and decision support systems shows, actual solutions to problems can often be of little use if it is not supported by a structured reasoned argument. The human, expert or otherwise needs evidence to support the conclusion (Turban '88).

It is this property that distributed systems lack. The nature of the hidden units in the systems is to adopt the required behaviour to ensure successful operation. Many localised network representations have been hand crafted, that is specially designed and developed, where the hidden nodes could be given clear functional interpretations. (Chapter three discusses some hand crafted neural networks). Given a generalised distributed scheme however the analysis has been limited. This thesis analyses the internal structure of neural networks and provides a suitable interpretive scheme.

Restricting ourselves to feed forward neural nets, several insights into the behaviour of hidden and output units can be gained. Viewing the global behaviour of the input/ output units we see the representation as Boolean transformations, (the analysis for noise affected, hard threshold devices is similar for perfectly trained nets). These transformations have a symbolic representation and logical rules can be constructed to represent them. These logical rules can be transformed to a neural representation and vice versa. The extraction of the rules from the net was introduced by Hinde('90) and this is discussed in chapter three.

Representational power of neural networks

Due to the distributed nature of many connectionist representations the question of representative power is ill defined. Hopfield gives the factor 0.15 * number of nodes, as the number of different classes that the Hopfield net can model, but this is a rather arbitrary figure. Lipmann('87) discusses the capacity of feed forward nets, suggesting the three layer model as the most general. He demonstrates further a limit on the number of intermediate nodes, based on an analysis of connected components in the input space. A figure of three times the input nodes, based on this argument and a convexity requirement is presented (each convex connected space requires at least three nodes).

Mirchandani and Cao('89) studied the role of hidden nodes in modelling decision regions and so presented some results relating the number of hidden nodes, dimensionallity of the problem and the number of decision regions required. This work is discussed with the new results on network size and topology in chapter seven, in which it is seen that for nets with a single output node, the hidden layer needs to be only as large as the input layer. This follows from the inherent planar nature of the points in multidimensional hypercubes. The results generalise showing how multioutput nets can still have limited hidden layers.

Tani et al('89) provide a different approach to minimising the size of a net used to model a problem. The method they employ is to include in the energy function a term that is related to the size and complexity of the net. This ensures minimising the energy function produces a minimised net representation. That is, after full convergence, the net discovered will represent the problem perfectly and have a minimal form. More problems of ensuring convergence exist here, but if these are overcome the representations formed would be topologically optimal.

Another application of the new energy function is in providing simplified rule representations of nets. Tani et al ('89), show how altering the energy function to allow over simplification of the net results in the discovery of simplified rules that approximately model the problem. This work relates to that of Hinde('90), showing how the hidden nodes may be associated with rules.

Connectionist symbol processing

The feedforward neural systems discussed so far have a representational power which is far below that of predicate systems. Recurrent networks offer the representational power of first order logical systems. The need to develop neural systems with the full power of predicate systems that can manipulate symbols has led to several different advances.

A limited symbolic representation exists in the coding of input/ output vectors as presented by Dolan & Dyer('87) but this is not true symbolic manipulation in the general sense. Hinton('90) and Pollack('90) present several ways in which recursive structures, trees, lists and hierarchies can be represented in neural systems. These techniques aim to extend the representational power of feedforward systems enabling them to execute general computation (Feldman et al '88 and Gallant '88).

Advantages of neural network systems

A neural network system offers several advantages over standard Von Neumann computational schemes. The first advantage is that of performance. A neural network system can be implemented in hardware offering great performance improvements over a conventional approach. The individual nodes in the network will be implemented as separate processors on a single piece of silicon. The activity of the nodes will pass from the inputs to the outputs in parallel producing extremely good performance.

The second benefit of a neural system is the existence of the training algorithms for implementing the networks. Any reasonably complex system requires a significant effort to design and build. Automatic training systems must exist to ensure suitable networks that produce the correct behaviour are constructed.

The types of training algorithm that exist vary considerably. The prescriptive methods of the Hopfield net ensure correct behaviour of the neural system, but in turn inhibits the representational power of the network. The backpropagation algorithm for feedforward networks provides a method for constructing neural networks that model a given training set.

A structured design and analysis of neural systems must be maintained if their behaviour is to be reliable (see chapter eight and nine for further discussion).

Neural networks have not, as yet, been used in general reasoning systems. This has been the case since no inference operator exists in neural systems. All the manipulations are at the bit level. This has led to the label of subsymbolic processing being applied to neural systems in general.

Neural techniques have been applied to specific reasoning problems, Hinton's ('89) knowledge base of family relationships is just one example. The knowledge, the family relationship between the set of people, is stored in the network systems by the relationships between the nodes and the information retrieved by the activity of the nodes. This network could correctly answer queries about the stored knowledge, that is act as an associative memory device. The network also correctly answered queries about relationships that it had not been specifically trained upon. That is some subsymbolic level of reasoning had taken place. The structure of this reasoning process requires further investigation. The complex interacting subsymbolic processes must be fully understood in order that the global behaviour of the network can be explained.

Neural network systems are trained on sample data, which are correctly stored in the net. The behaviour of the net in generalisation is not so predictable. The neural networks can learn to model any specified transformation (see chapter six and seven), but are only reliable over the training set. The generalisation properties are not well defined. A degree of well defined subsymbolic reasoning can be implemented by sufficiently constraining the network under consideration. The constraints induce a specified form of generalisation which is a manifestation of the subsymbolic processing.

Traditional logic based approaches are reliable. They are constructed incrementally and in general monotonically. The addition of each piece of new knowledge has a specific effect. Even nonmonotonic systems behave in a specified manner when more knowledge is added.

Neural systems generally do not behave this way. The system is only defined by the training set and is not reliable outside of this set. The different network representations that are produced when presenting the training set in a different order demonstrates the unpredictability of the systems. Adding a new piece of knowledge to the training set has similar unpredictable properties, often requiring a complete restructuring of the network to accommodate the new piece of information. A reliable approach to neural network design must exist if neural techniques are to be applied to general reasoning systems.

Summary

The basic properties of neural systems have been discussed, their training algorithms and representational properties. Their relevance to automated processes and reasoning systems have been examined. The notion that logical systems play an important part in ensuring neural network reliability has been introduced.

Chapter 2. Motivation of Study

Outline of chapter

The study that is presented in this thesis is motivated by the of lack of reliability of neural systems and their design. The questions of the design of the neural system, its interpretation and its behaviour are discussed. The lack of design tools available to the system implementor has led to ad hoc approaches (such as oversupply of nodes in the neural network and the training of the network over large data sets) in design. Neural network training has been well studied while that of network reliability and interpretation has not. This chapter examines the elements of neural network systems implementation that are important to the system designer. The properties of predictability and reliability are emphasised. The elements that must be investigated to ensure these properties are discussed, namely the network size and topology constraints as well as the internal function of the neural network. Having provided the motivation for the investigations of this thesis an outline of the thesis contents is given, highlighting the new contributions to the field.

Neural systems

The work in this study was motivated by the actual usefulness of the available techniques in neural system development. Neural networks have been used in many situations, from content addressable memories and distributed memories to constraint problem solving (Hopfield '84) and pattern recognition (Amari '67, Rumelhart et al '86 and Aleksander '90). However most work has concentrated on the properties of neural networks themselves and not on how to apply them to a given problem. This is seen from the work of Rumelhart et al('86), Minsky et al('69) and Rosenblatt('62). This work has been of little direct use to the designer of a neural network system.

The early work on the properties of neural network systems has led to the development of more general systems (Hopfield et al '86 and Hinton '89) which are applicable in a larger number of domains and so more useful to the systems designer. A general design methodology for such systems are not as yet widely used, but will be required if neural network technology is to become more successful.

Design development

Work on neural systems has proceeded in several directions, training, input representation and quantisation, and network structure and representation. These correspond to the three network properties that can be varied, the training procedure, the input transformation and the internal structure of the network.

To date training has attracted the most attention. This stems from the fact that any sufficiently large network structure is in theory capable of representing some given function or transformation (see chapter three and seven). Therefore the problem of applying neural network technology to a specific problem area reduces to a problem of training, having selected a suitably large network structure.

The attention to training has reaped many benefits. Almost all the advances in neural systems have been from the developments in training. The previous chapter described some current technologies all of which are based on a few standard network topologies and differ only in the sophistication of their training methodologies. These training methodologies are important, as without them designing neural systems would be difficult if not impossible.

The usefulness of neural network systems hinges on the input quantisation and representation as well as the network structure and representation that are available. The current techniques can all make use of the same input quantisations and representations. These are transformations of the input space that considerably simplify the problem space that is being modelled (Padaline techniques).

Little attention has been paid to the role that the internal structure and representation has when designing neural systems. It is often taken as a given property selecting one of the standard structures. Training then proceeds producing a network representation that will allow the training algorithm to converge. This approach neglects the role that structure and particularly the representation adopted has on the specific behaviour of the neural network. Predicting the behaviour of the network therefore is difficult if the representation adopted by the training scheme is not easily recognised and is perhaps unknown.

Usefulness of present techniques

Powerful training techniques exist in neural computing (automated learning via backpropagation), but they are not immediately applicable to a great many problems which are suitable for modelling by neural networks.

The four main areas of application have been;

i. Constraint problem solving

ii. Pattern recognition

iii. Control problems

iv. General neural computing

Early work centred on pattern recognition and constraint problem solving. Hopfield('84) made use of Hopfield nets to model constraints and relational systems. These systems allowed the modelling of specific problem spaces with neural techniques. Within the problem space the models behaved very well solving the particular constraint problem in question. Their limitations were in the transference to new related problems, which required new network solutions rather than minor variations of the networks that had already been constructed.

Pattern recognition in neural networks allowed the development of sophisticated vision systems. Fairly small recurrent neural network systems could be automatically trained to recognise and classify several different visual inputs (Aleksander '90). However they suffered from several problems which made them unsuitable for robust systems, these included pattern interference (that is linear combinations of stored patterns would also be recognised). Feedforward pattern recognition systems would not suffer from the same troubles but like all neural systems representational problems still existed. If no internal representation of the neural network existed then the behaviour of the network could not be guaranteed for input patterns that were not in the training set.

Neural networks have been used in the construction of control systems. From the work of Barto et al('83) and Zhang et al('91) to more complex safety critical systems (Miller et al '90 with the control of industrial robots), the neural network solution of control problems provides a significant area of application.

Predictability is a major concern in control problems. The behaviour of neural network controllers must be perfectly predictable if they are to prove practical. This is obviously vital when the question of safety critical control systems arise. Current techniques which rely on only a training set to specify the behaviour of a neural net are not suitable for these engineering applications. This is the motivation for the study in this thesis, the development of predictable and reliable neural network systems.

Areas of investigation

To date study has focussed on the theoretical basis of neural systems. Actual implementations have generally been lacking as several fundamental questions must be addressed before neural techniques can be applied to specific problems. This thesis addresses the questions raised by the attempt to implement neural systems.

When a neural network is being constructed, be it a theoretical project or a specific engineering implementation, several points must be addressed. The simple methodology in fig 2.1 highlights the three basic questions that must be examined when designing a neural network

i. Establish the number of inputs and outputs needed by the system.

ii. What is the initial internal structure of the network.

iii. What is the training algorithm that is to be adopted.

Fig 2.1 Simplified design methodology

Current work in neural systems has concentrated on the training aspect (part iii. of fig 2.1), the final part of the methodology for constructing neural systems. The first part of the methodology, the number of inputs and outputs will in general be specified immediately by the nature of the problem in question. If it is not clear, a system of analysis with the use of padaline techniques will identify important and relevant inputs and outputs to the system.

The greatest difficulty in pursuing this design methodology will centre around the internal structure that is initially adopted. The final internal structure of the net will be problem dependent and so the initial structure should be influenced by the problem in question. Most training algorithms do not manipulate the structure of the network itself (except via a form of network trimming) so an initial structure general enough to be suitable for all problems is used. A large fully interconnected network provides the most general option, but this still leaves the question of how large should the net be ? How many layers are required in the internal structure of the net and how large should the layers be ? This is the area that requires special attention. What is the smallest fully interconnected network that can model any problem with a specified number of inputs and outputs ? This question is addressed in chapter seven of this thesis.

Having provided a suitable initial internal structure for the neural network, training can proceed. A suitable training set must also be provided. This is especially the case with problems with a large number of input nodes as these would require an impracticably large number of training points to be fully defined.

Therefore new techniques must be provided to be able to systematically reduce the number of training points required to produce a suitable network that behaves correctly. This raises the question about how we can be confident about the behaviour of the net over input values that have not been used in the training phase of the network. Work is required to investigate the relationship between the structure and representation of the network and the behaviour that is produced. Is there a method for ensuring that a network is perfectly predictable over all points ? Is this related to the internal representation of the network ? What is the best internal representation for ensuring correct behaviour of network ?

Outline of thesis

The overall contribution of this thesis is in the study of neural network representation and behaviour and the presentation of a methodology for designing and constructing neural network systems. Part III of the thesis discusses the design methodology and the specific properties of networks that must be known when implementing this methodology. Its application to the design of reliable neural systems is examined.

The design methodology makes use of several techniques and properties that have already been well established (the feedforward network structure and the padaline linearising technique), as well as new techniques and properties developed in this thesis (for example, sandwich nodes which isolate independent regions which can be treated as knowledge atoms and results on the size of networks required to model specific problems).

The design methodology systematically presents the essential stages in constructing a reliable neural network system. The size and topology of a neural network must be known before the design can proceed. The number of nodes and layers required for the specific application must be specified. This question is extensively examined in this study and the results are presented in chapter seven. Boolean transformations are examined and results on the number of layers and size of the layers required presented. A new network topology the Loughborough Net is presented. This network topology exploits the parallel dependencies that exist in the nodes in the hidden layer of the network.

The analysis is extended to the case of real valued inputs in chapter eight. This draws on work by Huang et al('91), and Mirchandani et al('89), extending their work to the considerations of reliable network systems. Finally the methodology for engineering reliable neural networks is applied to the design of control systems. The Loughborough Control Net is presented in chapter eight. This is a new network topology suitable for implementing neural control systems. The Loughborough Control Net is applied to the design of a neural controller for a glue dispenser and presented in chapter nine.

Part II of this thesis discusses the properties of neural network systems and their representations. Interpretations of neural network representations are introduced. The interpretations of neural networks be it of a logical transformation, a rule system or another formal representation, is the only knowledge available about the internal structure and representation of the network. The importance of the interpretation is placed in the context that the belief about the behaviour of the network can only be based on this interpretation of the network. If this interpretation of the network is unreliable then the behaviour of the network will be unpredictable.

Chapter three examines the internal structure of neural networks and provides an interpretation of the nodes as Boolean transformations. This in turn provides an interpretation of the neural network representation as a system of rules and vice-versa. Understanding the internal structure of a neural network is essential for ensuring the reliable behaviour of the network.

In chapter four the structure of the internal representations of neural networks is discussed. The behaviour of the representations under various different learning algorithms is examined. Several specific problems are examined (parity in particular), to examine the success of the training mechanisms.

New representational dependencies are discussed in chapter five. These are the parallel and ghosting techniques which provide a computational approach to implementing sandwich nodes in networks. These can be viewed as the atomic knowledge elements which can be manipulated in the networks. Several experiments are described which show the various merits of different internal representational schemes. The computation merits of introducing dependencies into the internal representations is discussed.

The ability to make use of knowledge in the implementation of reliable neural systems is examined in chapter six. The sandwich nodes that are introduced here are an ideal representational scheme which are further developed and applied to control problems in chapter nine.

This thesis does not address general recurrent neural networks or symbolic neural computation. Both these areas are important fields of study in the development of general computational systems. The work in this thesis will aid the design of general neural reasoning systems, providing a framework in which the behaviour of neural networks can be fully understood. Chapter ten gives a more detailed account of the avenues that are opened by this thesis.

Summary

The motivation for the work in this thesis was analysed in this chapter. The need to understand neural network model's function and behaviour was emphasised. The lack of literature in this area was highlighted.

The structure of the thesis was outlined examining the themes of neural network representational properties, neural network interpretation and neural network reliability.

Part II. Properties of Neural Representations

Chapter 3. Interpretation of Neural Network Systems

Outline of chapter

The modelling of Boolean transformations by neural networks are examined in this chapter. Viewing neural networks as Boolean transformations gives an insight into their representational power. This is discussed further in chapter seven and eight.

An equivalence between bipolar neural networks and Boolean transformations is established. The modelling of standard Boolean functions with neural networks are examined and the Boolean representational power of single nodes are studied. The "object" definition of a Boolean transformation with a small number of inputs is presented. It is used as a tool to examine the structure of neural network models of Boolean transformations.

An interpretation scheme for transforming from bipolar neural network models to Boolean transformation models is presented. This is then extended to rule system representations.

The analysis of Hopfield networks as Boolean patterns of activity are examined and will aid analysis of the training techniques discussed in chapter four. Finally the interpretation of neural networks as models of training sets is examined. This gives insight into the neural network structure required to model the data in question.

Bipolar feedforward neural networks

A specific class of bipolar feedforward network are studied in this thesis, but the extensions and generalisations to other systems will be discussed. Throughout this section we will consider feedforward networks, with standard summing bipolar threshold units as the nodes (see fig 3.1). The output from node $Y_{k\,i}$ is given by the formula:

$$Y_{k\,i} = \textbf{threshold}(\Sigma^{n}_{j=0} w_{i\,j} \cdot Y_{k-1\,j}) .$$

$Y_{k\,i}$ is the output of the node i in layer k and $w_{i\,j}$ is the weight on the connection between the node $Y_{k\,i}$ and the node $Y_{k-1\,j}$ and $Y_{\cdot\,0}$ is identically 1 and is known as the bias of the node.

threshold is the threshold function (fig 3.2) defined as:

threshold(X) = +1, for all X > 0,

threshold(X) = -1, for all X < 0.

Therefore whatever the input to any particular node the output will always be either +1 or -1. This type of network can be easily generalised to those using a sigmoid function with a large derivative at the zero point (see fig 3.2c). That is , for a sigmoid function defined as:

sigmoid(X) = (2/(1 + exp(-X/T)) - 1)

where T is the temperature, if T is small the derivative of **sigmoid**(X) will be large at X = 0. The temperature T is taken from the analogy with the Bolzmann nets (Hinton '89 and Rummelhart et al '86) which generate the output values stochastically based on the output of the sigmoid function.

A node with a threshold unit will always have bounded output values. If the unit is thresholded in a bipolar manner the output values will be constrained such that $-1 \leq Y_{ki} \leq 1$, see fig 3.2 a,b & c. Given a sigmoid threshold function with a low temperature factor T or conversely a large input value X_{ki} then the output values of the node can effectively be constrained to $-1 \leq Y_{ki} \leq -1 + \beta$ or $1 - \beta \leq Y_{ki} \leq 1$, where β is a small positive constant. Therefore given a network with a low temperature, ensuring high input values will effectively guarantee Boolean decision values. That is the output from each node is either +1 or -1. When a network has converged after training, the nodes in the network essentially behave as Boolean units (see chapter four), all the weights have been suitably reinforced to produce large inputs.

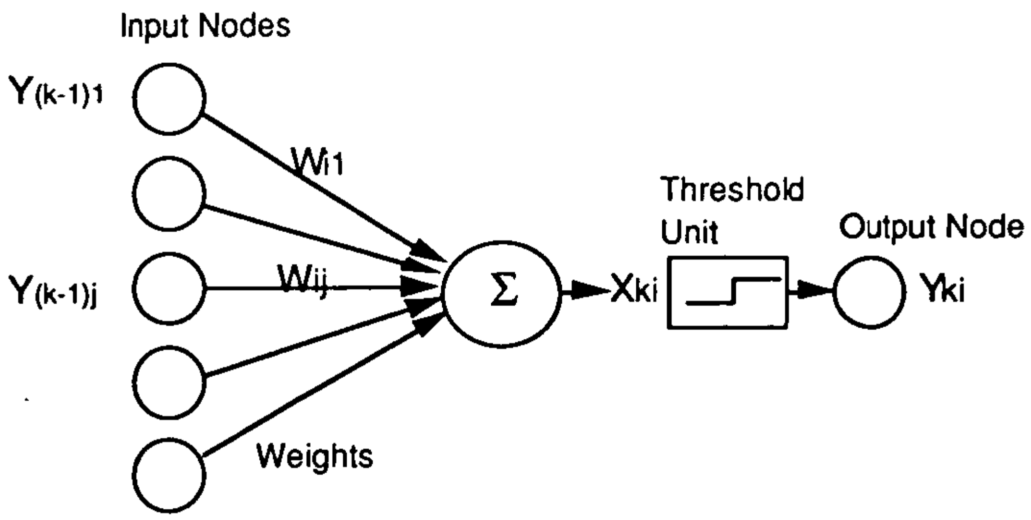


Fig 3.1 The connection scheme between layer k-1 and a node on layer k in a feedforward network

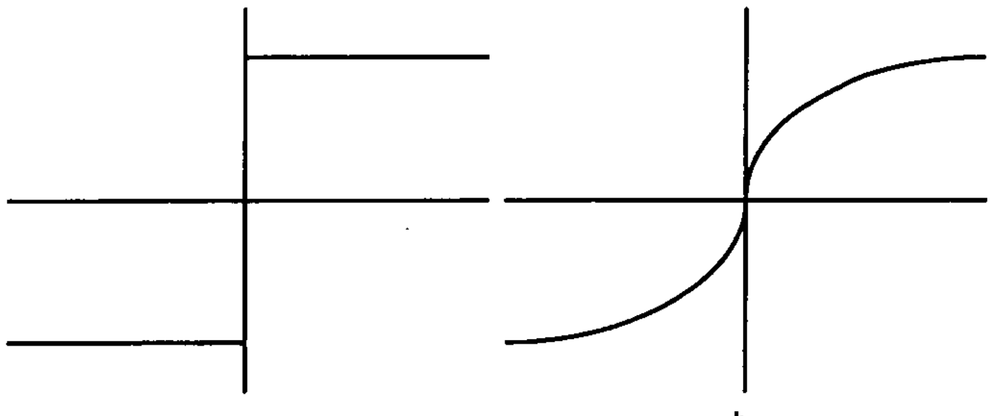


Fig 3.2 a. Threshold function **threshold**, b. Sigmoid function **sigmoid**

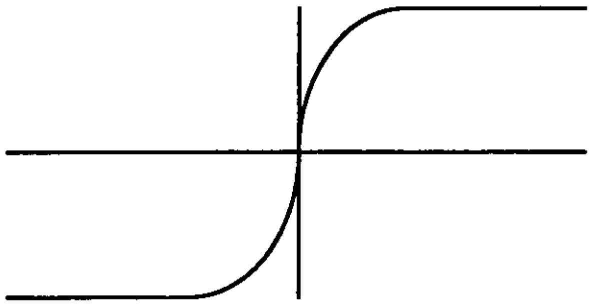


Fig 3.2 c. Sigmoid function with a low temperature

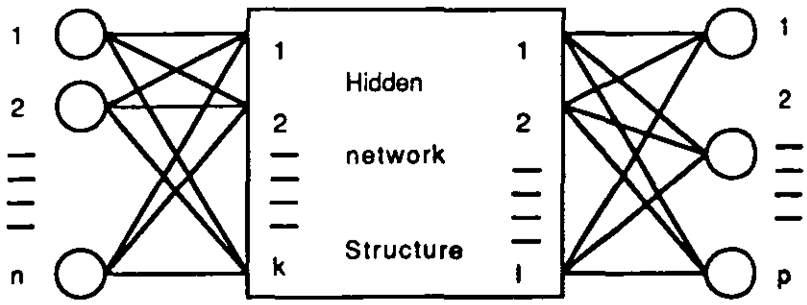


Fig 3.3 Structure of a general neural network. There are n input nodes, p output nodes. The activity feeds forward from input nodes to the output nodes

The function f: $B^n \rightarrow B^p$, a Boolean transformation from n inputs to p outputs, can be modelled by a bipolar neural network with n inputs and p outputs. Fig 3.3 shows a general neural network with a multilayer hidden network structure. The intermediate network structure that is required to model the given mapping is discussed in generality in chapter seven. The properties of individual nodes and their ability to model Boolean functions is examined in this chapter.

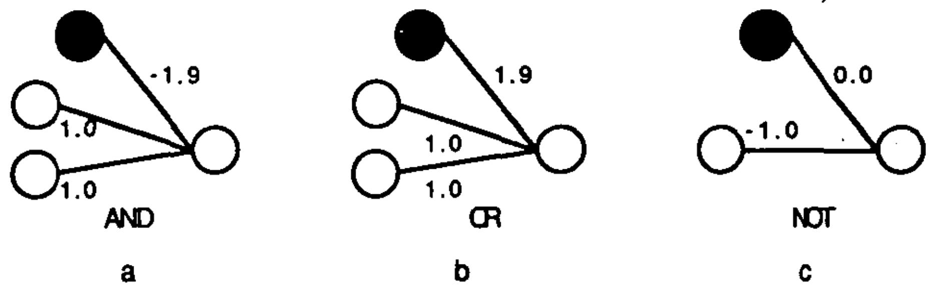


Fig 3.4 Single node representations of; a. AND, b. OR, and c. NOT. The bias nodes are shown as solid circles, the values of which are identically 1.0

Any Boolean transformation can be modelled by a neural net. This is the case since we can give a formula of any Boolean transformation using just the operators AND, OR, and NOT, which in turn can be modelled by a single node each (see fig 3.4 a, b, c) and placing the formula in disjunctive or conjunctive normal form. The standard logical AND and OR operators for any number of arguments can be modelled by one layer of weights, that is a single node. As seen from fig 3.5 a, b, the AND is constructed by a large negative weight and the OR by a large positive weight on the bias line that can only just be overcome by all of the arguments being true (+1) or false (-1) respectively. The NOT operator is represented by a negation of the value of the weight connection and so does not add any extra layers to model.

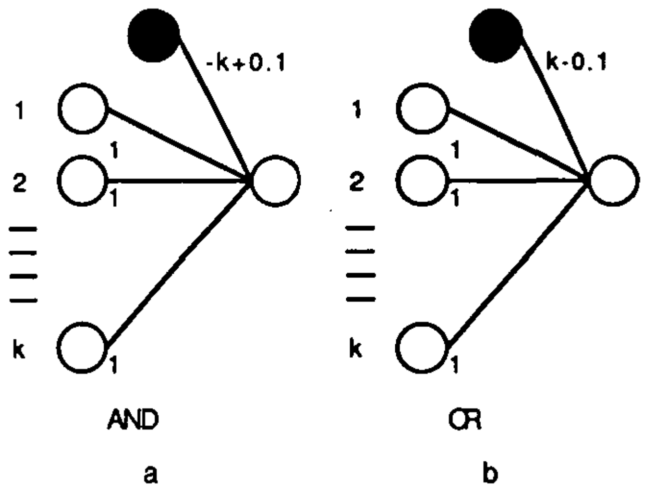


Fig 3.5 Single node representations of a. k input AND and b. k input OR transformations

Therefore the neural network can be constructed from the Boolean formula of the transformation with node units that represent the atomic Boolean operations. Huang et al('91) presents a different scheme for producing a neural representation of a Boolean transformation given the training points to be modelled. This is achieved by constructing hidden nodes that isolate the separable elements of the training set. In general these techniques do not produce optimal representations. That is, networks that use the minimal number of nodes that are required to model the problem in question. This is largely due to the fact that strictly Boolean operators are a subset of those that can be implemented by a single neural operator. This is easily demonstrated by taking the two extreme cases of the k input AND and the k input OR operators shown in figure 7. The OR node represents the truth of the statement onenode(on), that is there is at least one node firing +1 in order for the output to be +1. By reducing the bias weight by 2.0 we require any 2 nodes to be on in order for the output to be +1 giving a modal operator twonode(on). There are many of these operators derived solely by altering the bias weights and keeping the input weights fixed at 1.0. By altering the input weights we can derive a further large class of operators. Although these operators are more general than the class of Boolean operators they cannot represent more statements than those representable by an arbitrary collection of Boolean operators, however they are more economical in their representation.

Object definition of Boolean transformations

Neural networks with a single output node are examined in this section. This simplification aids the analysis considerably although some of the results which are immediately applicable will be extended to the multiple output case. We can view any particular transformation with n input nodes and one output node as defining an object in an n dimensional Boolean space, where if $f(\mathbf{x})= +1$ then that point is in the object, i.e. it is of interest, while if $f(\mathbf{x})= -1$ then that point is outside of the object, it is not of interest.

The nodes in the hidden layer represents a hypersurface in the n dimensional space that separates the space into two regions. One where the node gives a value +1 and the other side of the hypersurface where the node gives the value -1. This idea is used to produce diagrams of specific transformations. The input space is shown as corners of a hypercube. When $f(\mathbf{x})= +1$ the point is shown by a filled circle while when $f(\mathbf{x})= -1$ the point is shown by a empty circle. Exclusive OR in two dimensions is shown in Table 3.1, and its object definition in fig 3.6a. These object representations will reduce the need to give full input/ output definitions for particular transformations under consideration. Annotations with planes representing the hidden nodes will remove the need to give the network representations. The object representation will show both the transformation definition and the networks that model it.

		B	
	XOR	+1	-1
A	+1	+1	-1
	-1	-1	+1

Table 3.1 Table of values for the transformation XOR shown in fig 3.6a

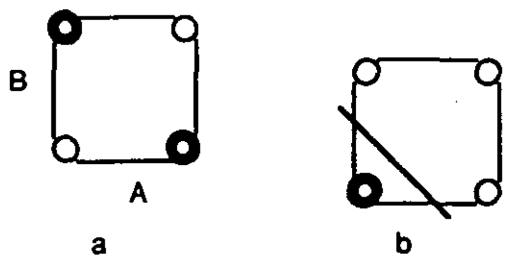


Fig 3.6 a. Object representation for XOR, b. Object definition of not(or) showing the node line that models the problem

The node defined by the bias -1, and two weights -1 and -1 (represented as node(-1,-1,-1)) defines the Boolean function not(or). The object definition of this function and the line that represents node(-1,-1,-1) is shown in fig 3.6b.

As stated above, any node can be represented as a Boolean transformation. Its formal representation may be particularly complicated, nevertheless it is a Boolean formula. Given a node we can convert it to Boolean form by the following technique.

Given a node with its bias value and weights, say $(b, w_1, w_2, \ldots, w_n)$;

if $|b| > \Sigma^n_{i=1} |w_i|$, then node value is True if $b > 0$, or node value is False if $b < 0$, and similarly for all the weights;

if $|w_1| > |b| + \Sigma^n_{i=2} |w_i|$, then node value is input_node_1, if $w_1 > 0$, or node value is input_node_2, if $w_1 < 0, \ldots,$

if $|w_n| > |b| + \Sigma^{n-1}_{i=1} |w_i|$, then node value is input_node_n if $w_n > 0$, or node value is input_node_n if $w_n < 0$.

Otherwise node is $(b + w_1, w_2, .., w_n)$ o r $(b - w_1, w_2, .., w_n)$, where $(b + w_1, w_2, .., w_n)$ and $(b - w_1, w_2, .., w_n)$ are two nodes with one less input node than the original node.

The above provides an iterative scheme for converting all neural network nodes to Boolean formulae. A complete neural network can similarly be converted into a single large Boolean transformation by converting all the nodes in the network to Boolean representation.

Example

Given the node D in fig 3.7, we can derive a logical definition of its function.

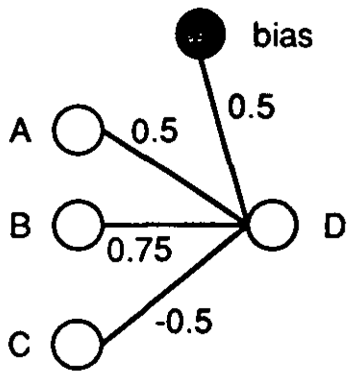


Fig 3.7 A node with three inputs

node(bias, A, B, C) has no weight such that $|w_j| > \Sigma^3_{i=0, i \neq j} |w_i|$, therefore set A= +1 and A=-1,then;

node(bias, A, B, C) = (A and node(bias+0.5, B, C)) or (not(A) and node(bias-0.5, B, C)).

node(bias+0.5, B, C) has no weight such that $|w_j| > \Sigma^2_{i=0, i \neq j} |w_i|$, therefore set B= +1 and B=-1,then;

node(bias+0.5, B, C)= (B and node(bias+0.5+0.75, C)) or (not(B) and node(bias+0.5-0.75, C)).

node(bias-0.5, B, C) has weight B such that |0.75| > |0| + |0.5|, therefore node(bias-0.5, B, C) = B.

node(bias+1.25, C) has weight bias such that |1.75| > |0.5|, therefore node(bias-0.5, B, C) = +1.

node(bias-0.25, C) has weight C such that |0.5| > |0.25|, therefore node(bias-0.25, C) = not(C).

Therefore D = node(bias, A, B, C) = (A and (B and +1) or (not(B) and not(C))) or (not(A) and B)) = ((A and B) or (not(B) and not(C))) or (not(A) and B)).

The Boolean formulae for the nodes can be considered to be rules and the neural network a system of rules. The equivalence of the node to the Boolean transformation is interpreted as an implication operation. Namely the logical equivalence;

D = ((A and B) or (not(B) and not(C))) or (not(A) and B)) is given as;

D <- ((A and B) or (not(B) and not(C))) or (not(A) and B)), with the usual definition of implication. Normal node execution performs the modus ponens reasoning scheme. There is no natural and simple network execution strategy that will perform modus tollens.

This provides a natural transformation between rule systems and neural networks. Due to the feedforward nature of the neural networks, no recursive rules can be implemented in the neural system.

Example

The network given in fig 3.8 can be interpreted as the rule set;

D <- f1(A,B,C), E <- f2(B,C), F <- f3(A,C), G <- f4(D,E), H <- f5(D,F), where the functions f1-f5 are Boolean transformations.

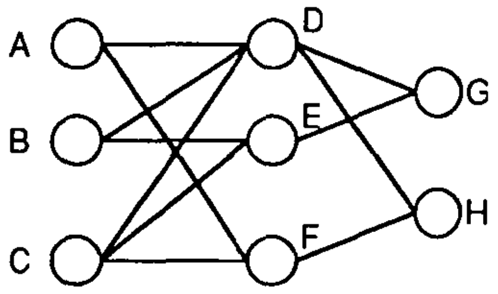


Fig 3.8 A neural network that can be interpreted as a rule system

Boolean models of Hopfield network behaviour

The nodes in a Hopfield recurrent network can be modelled by Boolean activities. This is natural if we have hard limiting logical devices, but also applies to sigmoid thresholded systems in which the network has converged to a stable state. In this case the stable values are Boolean patterns and so the representation and behaviour can be analysed accordingly. The stable state patterns are stored over a fixed number of nodes and so the behaviour of the Hopfield network can be studied by analysing the matrix of activity patterns over the different nodes. This is illustrated in fig 3.9 for the stored pattern in table 3.2. The recurrent network structure that models this problem is shown in fig 3.10.

pattern \ node	0	1	2	3
0	-1	-1	-1	-1
1	-1	-1	1	1
2	-1	1	-1	1
3	-1	1	1	-1
4	1	-1	-1	1
5	1	-1	1	-1
6	1	1	-1	1
7	1	1	1	-1

Table 3.2 Matrix of node activity over the training set

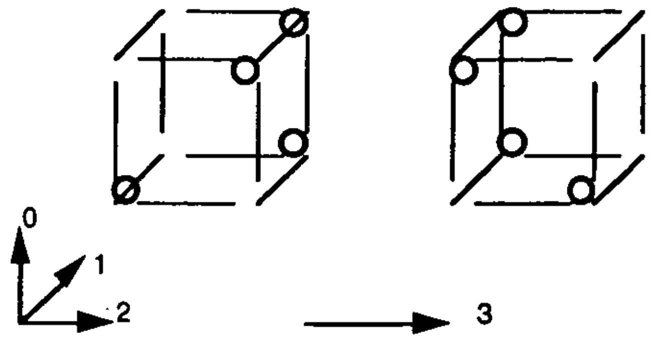


Fig 3.9 Plot of the training points shown in table 3.2

Whenever we are interested in the behaviour of a particular node, over the various input patterns (that is the stored patterns), the relevant column in the activity matrix provides the required information. Since the pattern of activity of different nodes are distinct (if this were not the case they could be amalgamated (see chapter seven) and treated as a single node), the node behaviour can be correlated against the other distinct nodes. The correlation values of the node activities are important when we come to examine the representational properties and training schemes of Hopfield networks (see chapter four).

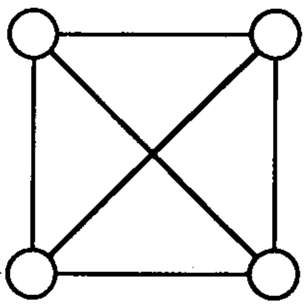


Fig 3.10 Hopfield network used to model the problem in table 3.2

Training sets and interpreting neural networks

Throughout this study, neural networks are examined on their ability to model a specific transformation. In the recurrent Hopfield network regime there are no specific input and output nodes as such. A number of patterns are stored by these networks. The network truly represents the patterns that are stored over the nodes and not some specified transformation from input nodes to output nodes.

With this motivating example we can view feedforward neural network systems via the training sets that they are required to model. Given these conditions the neural system need only model the training set in question and its performance over any other point is ignored, it is not drawn into question. This contrasts with the transformation approach to feedforward systems, where the network has to model a specific transformation which is defined over all the input space.

The usefulness of this approach is in the number of different transformations that can possibly model a small training set. A small training set specifies output values for only a few input points in the training set and so does not constrain the values of the outputs on the remaining input points. Therefore many underspecified transformations will be capable of modelling the training set in question.

A training set has the following properties. The training set is a set of arbitrary binary data points and so is not dependent on the network architecture under consideration. The data itself need not be viewed as specific input and output values but should be viewed as activity over specified nodes. In the feedforward neural network environment, the structure of the network forces an interpretation of input and output nodes onto the various nodes in question.

A training set can be viewed to be a definition of the space of interest, the points that are in the training set, and that not of interest, the points that are not in the training set. Given the specific training set we can predict whether a particular network structure can model that training set or not. This is further examined later in this chapter.

pattern \ node	Input 1	Input 2	Output
0	-1	-1	1
1	-1	1	-1
2	1	-1	-1
3	1	1	1

Table 3.3 The exclusive or transformation from two inputs to one output

A feedforward architecture has explicit input and output nodes (Table 3.3), yet the training set can still be viewed as a data pattern explicit from the input output structure (Table 3.4). The training set are then just bit patterns that have to be stored in the network. With the unbiased data points, we can make any input/output decisions required to model the data with the given architecture. Effectively, we can break away from viewing the training set as a predefined input and output structure, but can choose whichever nodes most effectively perform the function of an output node.

pattern \ node	0	1	2
0	-1	-1	1
1	-1	1	-1
2	1	-1	-1
3	1	1	1

Table 3.4 Training set for the exclusive or problem with two inputs and one output

The properties of the network architecture chosen will dictate whether the training set can be represented or not. The property of the McCulloch and Pitts neuron which can only distinguish linearly separable sets provides the representational limitations of neural networks. Training sets as arbitrary patterns arise naturally in recurrent network structures such as the Hopfield network. The properties of Hopfield networks and their abilities and limitations at storing bit patterns are discussed in chapter four.

For a feedforward network to model a training set an explicit output must be defined. This is the output node specified by the network architecture of the feedforward network. The simplest method of defining this output node is to provide the value +1 if the given input point is in the training set and -1 if the given point is not. That is the network works as a metalevel pattern recogniser, rather than as an implicit input output transformation unit.

pattern \ node	0	1	2	3
0	-1	-1	-1	-1
1	-1	-1	1	1
2	-1	1	-1	1
3	-1	1	1	-1
4	1	-1	-1	1
5	1	-1	1	-1
6	1	1	-1	-1
7	1	1	1	1

Table 3.5 Metalevel definition of Exclusive or with two inputs and one output, the final node provides a decision as to whether the pattern over the other nodes is a stored pattern or not

When a data point is presented to the network, the activity is passed through the network, resulting in the final output of +1 or -1, depending on whether the presented pattern was part of the training set or not. The main problem with pursuing this approach is the inefficiency of the network structure adopted. Many extra metalevel training points must be provided to fully define this pattern recogniser. This is illustrated by examining the "exclusive or" example. Table 3.5 illustrates all the points that must be stored in this metalevel network system, when the original training set was a large factor smaller (table

3.4).

To fully exploit the power of the feedforward representation, we must employ a network that has at least one explicit output node. That is an output node that behaves as one of the nodes in the training set. For such a node to exist, the training set must be closely examined. For a node in the training set to be an output node, it must be uniquely defined for all the patterns in the training set over the other nodes in the training set. This is illustrated by the example in table 3.6a, where any single node can act as an output node. In table 3.6b, node A and node B can not act as output nodes, since if A was an output node, we would have the transformation B, C -> A : (-1,1) -> 1 and (-1,1) -> -1, which is inconsistent, and similarly with B as an output node. The only possible consistent output node is C.

pattern \ node	A	B	C
0	-1	-1	1
1	-1	1	-1
2	1	-1	-1
3	1	1	1

a

pattern \ node	A	B	C
0	-1	-1	1
1	-1	1	1
2	1	-1	1
3	1	1	-1

b

Table 3.6 a. Training set where any node can act as the single output node, b. Training set where node C is the only possible consistent output

Multiple output nodes

Given more than one possible output node, we can select any one to act as the output node of the system. Having selected one output node, it may be possible to select more output nodes without any inconsistencies arising. We continue this process until no more output nodes can be selected. In Table 3.6a, any node can be a single output node, but no two together can act as output nodes since inconsistencies arise. For transformation A -> (B,C) we have 1 -> (1,1) and 1 -> (-1,-1) and similarly for output nodes (A,B) and (A,C).

pattern \ node	A	B	C	D
0	-1	-1	1	-1
1	-1	1	-1	-1
2	1	-1	-1	1

Table 3.7 A Training node that explicitly allows multiple output nodes

Table 3.7 shows an example where any node can act as a single output node, but only (D, A or B or C) can be double output nodes. That is, (B,C) can not be double output nodes since (A,D) -> (B,C) have the training points (-1,-1) -> (1,-1) and (-1,-1) -> (-1,1), which are inconsistent.

Summary

This chapter examined the role of Boolean representations of neural networks. The equivalence between converged neural networks and Boolean transformations brings the connectionist techniques into the realm of reliable and predictable systems. These ideas are further developed in the following chapters.

Neural networks were examined as model of training sets. The training sets admit an interpretation as a transformation which can then be modelled by feedforward systems. This notion of neural models of training data will be used to construct a measure of neural network reliability and predictability in chapter four.

Chapter 4. Learning and the Behaviour of Internal Representations

Outline of chapter

Backpropagation is used to train feedforward neural networks. The training algorithm executes iterative hill climb to minimise the defined error function of the system. Perfect steepest gradient descent schemes would force the system into the nearest local minima. This is undesirable, so varying iterative schemes are examined. These hope to avoid the local minima and converge to the global minima. This chapter provides some insight into the experimental result on neural network convergence.

Training is examined as the problem of modelling data sets with neural network systems. Different training schemes are discussed, outlining the role of temperature, learning rate and momentum terms in the backpropagation training algorithm.

The local properties of the training algorithms are examined. The effect of training on the weights and node activity are examined. The role of temperature, learning rate and momentum terms in avoiding problems of local minima are examined.

The training algorithms developed for Hopfield nets (Aleksander et al '90, Hinton '89 and Rumelhart et al '86) are examined. The training of hidden nodes in recurrent networks are examined particularly the Markov chain models. It is shown that specific cases exist where the hidden nodes can be trained using a one pass Hebbian training rule.

The training of networks over reduced training sets are examined and the role that they play in the reliability of the neural network is discussed. Two measures of reliability are introduced. One based on the uncertainty in the transformation being modelled and the second based on the reliability of the behaviour of the neural network.

Training Neural Networks

In order to construct useful network representations of transformations, a specific methodology must be adopted. The following three elements are required;

i. the initial network structure,

ii. the set of data on which to train the net, and

iii. the training algorithm that is needed to adapt the initial network structure.

More broadly we can view this as follows;

i. the initial knowledge of the system.

ii. the new knowledge to be imparted, the training set, and

iii. the training methodology.

When implementing a neural network for a specific problem it must be trained. Training a neural network is a matter of manipulating the behaviour of the network. Once the behaviour corresponds to that which is desired, training can cease.

The system requires a specific behaviour, that is a specified transformation from input to output. A specific Boolean transformation may exist, a rule system or another formula for the desired behaviour. This behaviour will be defined over a given subset of the input space. Training will proceed over this subset of the input domain. The neural network is said to have converged when the training set is correctly modelled. The correct behaviour of the neural network is only guaranteed over the training set, therefore the total behaviour of the neural network is only guaranteed if the training set is the whole input space. That is neural network training should be viewed as a convergence to correct behaviour over the training set. Correct training of the neural network can be viewed as the representation of the given Boolean transformation, rule set or another specified formula of behaviour.

Available knowledge

When modelling an arbitrary transformation there are some pieces of information available, we have some knowledge of the system. The most basic knowledge is that of the number of inputs and outputs, which is defined by the nature of the transformation being modelled. This is available even if the actual specifics of the transformation considered is unknown. Any binary valued transformations have well defined input and output fields, while continuous real valued input and output fields can be digitised to a prerequisite degree of accuracy. Nothing is lost by specifying this digitisation of the input fields since all neural systems essentially deal with digitised data after the first layer of nodes.

Given the limit of one hidden layer in a network as discussed in chapter seven, we specify that the Boolean neural networks we construct have no more than three layers of nodes. The limit of the same number of hidden nodes as input nodes for each output node as also discussed in chapter seven provides us with the number of hidden nodes that the network will have. This will be the maximum number of nodes and layers that we will require, whatever the actual transformation being modelled. The nodes in successive layers are fully interconnected. Given this topological limit on the network we can proceed to train it to behave correctly.

When no knowledge about the actual transformation being modelled exists, the above network structure is the best that can be constructed. The value of the weights in the network are given by the specific knowledge that may exist about the transformation being modelled. If the initial knowledge about the transformation is nil, then this will be reflected in the weights of the network. When no information exists small random weights are used since if they were all zero, the training algorithms would not distinguish them from the case where no links exist. The experiments of appendix A1 show that the initial weight values of the neural network have a great influence on the success of the automated training algorithm.

If knowledge about the transformation exists, this is the stage at which it should be included. If the transformation is known to be symmetric about an input, this would be reflected in the weights. If a network representation of a subspace of the transformation is known, this would be a stage to include it in the new representation. Improvements in the training phase will be gained by the use of this knowledge. An experiment is discussed in appendix A4 in which the training performance of neural networks initialised with different information about the transformation being modelled is examined.

Training examples

The training examples are used to to train the network, they are a subset of the total input space. The initial configuration of the network will in general not correctly represent the training set. The training algorithm adapts the network to the point where it maximally represents the training set. If the algorithm does not become stranded in a local minimum, the network produced after training will fully represent the training set.

Unless the training set is full, that is, it is the complete input space, it cannot be guaranteed to model a specific transformation. This means that points in the input space that are not in the training set will behave in a manner dependent on the initial configuration of the net and the training algorithm employed, and not the particular transformation in mind.

We can see that this is the case from the following analysis. Given a network with a specific weight configuration we can train it on a subset of a transformation say A and also on a subset of a transformation B. If $A(t)= B(t)$, for all t from the training set, then the two networks that have been trained will be identical. Now if A and B are distinct transformations then there will exist a point x say, outside the training set for which $A(x)\neq B(x)$. The network representation of A and B will both produce the same output for the input x and so one of the network representations will be incorrect for this input.

Therefore predicting the generalisation properties of networks are almost impossible. More knowledge about the network representation produced must be employed before we can predict its behaviour over points not in the training set.

Training methodologies

The training methodology most commonly used for feed forward networks is that based on backpropagation. The procedure for investigating learning via backpropagation involves examining the behaviour of the net over the training set and seeing how much the actual behaviour differs from that of the desired behaviour. All the weights in the network are updated in a direction and by an amount that minimises this difference. The details of the algorithm are discussed below.

Backpropagation

The error in the output for a particular input x, E(x), is given by the formula;

$E(x) = D(x) - R(x)$,

where D(x) is the desired output for the input x while R(x) is the actual output for the net. The update that is applied to each weight in the net is proportional to its effect on the error function, namely, $\Delta w = -\mu \, dE/dw$, where μ is a positive constant called the learning rate.

The backpropagation may proceed iteratively over the input values, that is the weights are adjusted after each example from the training set is taken. A different approach is to calculate a total error function over all the training set. This second method would behave in a truer gradient descent manner, but as discussed below, this in itself is not an advantage when the search space has many points of local minima. Therefore for the most part, systems that adjust the weight space after each pass of the input examples have been implemented.

It can be seen that backpropagation instigates an error gradient descent in weight space. If it were a true steepest gradient descent procedure, then it will almost certainly get stranded in any local minima that exist. This is the case, since in a network which is near a local minimum, the steepest gradient will be towards that minimum. On reaching the minimum the training algorithm will not be able to move away since the gradient will be zero at that point. For real applications we must utilise a training algorithm that overcomes these problems. Several strategies already exist.

Temperature, learning rate and momentum terms

In order for the backpropagation algorithm to execute true steepest gradient descent any update in the weight space must be infinitesimal. In any real implementation this is impossible.

The first factor that affects the behaviour of the backpropagation algorithm is the learning rate. This is the proportional constant that influences the magnitude of the update that is carried out on each iteration. If the learning rate is too high, then the network will oscillate between nonoptimal states. If it is too low then convergence will take many iterations, which is undesirable. (See appendix A2).

There is a close relationship between the learning rate adopted and the temperature of the network system. The temperature is the proportional constant applied to the input in the sigmoid threshold function. The node formula is given by Output= sigmoid($\Sigma w_i x_i$), where $w_i x_i$ are the weighted outputs of the nodes from the previous layer. The sigmoid function is given by the formula;

sigmoid(x)= 2/(1+exp(-x/t)) - 1, and

dsigmoid(x)/dx = 2 exp(-x/t)/ t $(1+\exp(-x/t))^2$.

The update formula for the weights w can be given in the form;

Δw= -μ dE/dw = -μ dE/dv.dv/dy.dy/dw, where v= sigmoid(y), y= $\Sigma w_i x_i$,

so Δw= -μ. 2(exp(-x/t)/ t).$(1+\exp(-x/t))^2$.(dE/dv.dy/dw).

Therefore from the formula above we see that the term 1/t behaves like a learning rate parameter. That is, by decreasing t we can effectively increase the learning rate. (See appendix A3).

The final method by which standard backpropagation algorithms deviate from steepest gradient descent approaches are with the use of momentum terms. If a steepest gradient descent procedure gets stranded about a local minimum the search is unable to proceed further, the algorithm oscillates about this minimum. To inhibit this property, momentum terms are applied to the update procedure. The function is to reduce oscillation and ensure that the search proceeds in a purposeful direction.

The momentum terms are implemented by applying a proportion of the previous update to the present update. That is the weight update formula is given by;

$$\Delta w_{t+1} = -\mu \; dE/dw_t + \beta . \; \Delta w_t$$

as the iterative cycle continues the effect of the momentum term diminishes if $\beta<1$. That is the momentum term is only effective for a few cycles after it is initialised, it decays.

Properties of the backpropagation algorithm

Training a neural network system with the backpropagation algorithm is a balance of interrelated elements. The combined action of the nodes in the different layers of the network produce the specific behaviour. The training algorithm coordinates the perturbation of the weights and nodes such that the total behaviour of the network over the training set converges to that which is desired. The training algorithm is examined in detail, emphasising the specific elements that aid optimal convergence.

Training an individual node

Given a particular output node (fig 4.1) we can define an error function on the node for each training example as;

$E = (O_d - O_a)^2$, where O_a is the actual output from the node while O_d is the desired output.

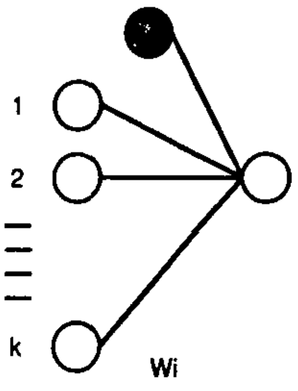


Fig 4.1 A single node with several inputs and one bias weight

The update formula for the backpropagation algorithm is;

$\Delta w_i = -\mu\ \partial E/\partial w_i$, where E is the error function of the network, $-\mu$ the learning rate, a small constant and w_i the weight values. The iterative update of the weights is such that the error function is minimised.

The first point of interest about the update rule is that when a point is correctly classified, no change in weight is made. Only incorrectly classified points contribute to the learning. This is desirable since if all the training set is correctly modelled by the network, no update in the weight values would be required. One disadvantage is that there is no positive training factor. That is all the points that are correctly classified will not contribute to actively maintaining the structure of the network. There is no resistance force from the correctly classified points to changes in the weight space. The few incorrectly trained points will provide all the forces for developing the network.

If the incorrectly classified points contribute constructively in training then the algorithm will converge rapidly. Fig 4.2a illustrates this point.

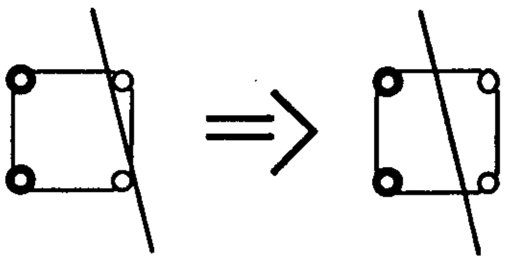


Fig 4.2 a. Constructive training

In destructive training the incorrectly classified points force the representations to either oscillate or diverge to such an extent that originally correctly classified points become misclassified. Fig 4.2 a & b illustrate these points.

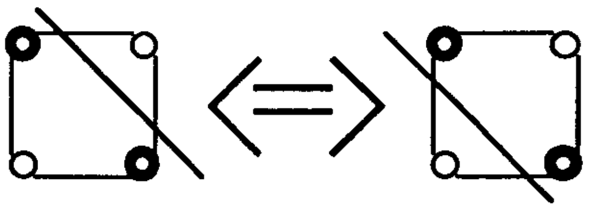


Fig 4.2 b. Destructive training 1) oscillation

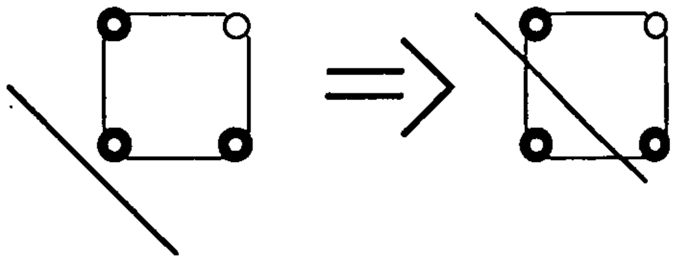


Fig 4.2 c. Destructive training 2) divergence

The simple examples above illustrate cases where the distinction between constructive and destructive learning are clear. Given systems with many more inputs and incorrectly classified points, changes that reclassify a few points either correctly or incorrectly, can not so easily be termed constructive or destructive learning. Only the global behaviour of the algorithm in correctly classifying different numbers of points can allow this judgment. This can be seen in the results of the experiments described in appendix B. During backpropagation training, many iterations produce an increase in the sum squared error measure. Destructive learning has occurred at this point. Often this destructive learning is advantageous as it allows the present representative structure to be broken, so allowing a more favourable start point from which to converge.

Local learning

The specific effect of an incorrectly classified point on the training algorithm is a significant point of interest. The global activity of the training algorithm will depend on the interactions of these micro activities. By examining the effect of an incorrectly classified point on different nodes we will be able to gain an intuitive idea of the effect of the point on all the different nodes in a particular network representation.

Given a node threshold function $f(\Sigma w_i Q)$, we have the node weight update formula;

$\Delta w_i = -\mu(O_d - O_a)(df(x)/dx)O_i,$

The term $(O_d - O_a)$ depends on how badly the training point is classified. Assuming we have just Boolean transformations, (a hard threshold function is employed), then $(O_d - O_a) = 0$ if just Boolean transformations, (a hard threshold function is employed), then $(O_d - O_a) = 0$ if the point is correctly classified and $(O_d - O_a) = \pm 2$, if incorrectly classified. Similarly $O_i = \pm 1$. All these elements provide a component as to the direction of the change that is most suitable to model the training point in question. The magnitude of the change is dependent on the term df(x)/dx, which for a hard threshold function is almost zero (small) everywhere except when x is almost zero, where df(x)/dx is very large. This gives the result that for training points where $\Sigma w_i Q$ is not almost zero, the magnitude of Δw_i is dependent on the learning rate, that is this term must be adjusted for optimal performance.

Now consider the case where we have outputs in the range $|O_a| \leq 1$, that is we have soft threshold functions such as the sigmoid function with a high temperature term. These conditions give the following constraints, $|(O_d - O_a)| \leq 2$, $(df(x)/dx) = 2\exp(x/T)/T(1 + \exp(x/T))^2$, which is shown in fig 4.3. The important points to note are that (df(x)/dx)>0 and that it attains its maximum at x=0. (See fig 4.3).

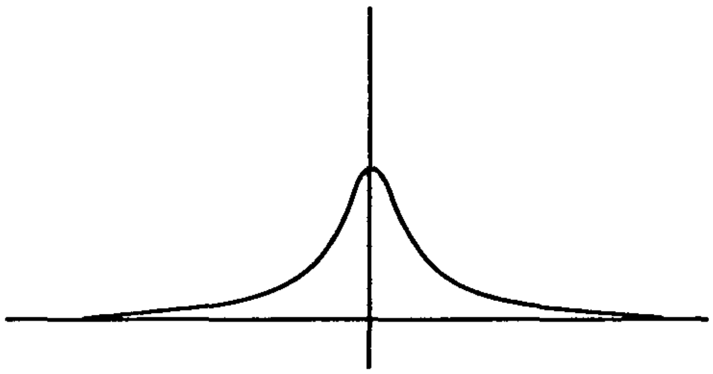


Fig 4.3 The differential of the sigmoid function

With these criterion we can see that the magnitude of the weight update depends on how badly classified the point is and how close $\Sigma w_i Q$ is to zero. The measure $\Sigma w_i Q$ is a function of how close the training point is to the hyperplane that the node represents. Therefore the closer the training point to the node plane the greater its effect on the training. If $|(O_d - O_a)| << 1$, then effectively the point is correctly classified. However, since the term is not zero there is still a contribution to the training algorithm. This is an advantage over the strict Boolean case, since even essentially correctly classified points have an effect on the training algorithm.

If we now consider the case where the input values can range over the values, $|O_i| \leq 1$, we have a different effect manifesting itself in the training algorithm. The activity of the input nodes to the node have an effect not only on the direction of the update of the weights, but also its magnitude. The greater the activity of the particular input node, the greater the update in the weight value. This means that if the input node provides hard evidence, that is $|O_i| \approx 1$, it has the greatest effect on training, while if $|O_i| \approx 0$, very little training is carried out.

Training a two layer subnetwork

Refer to the annotations of fig 4.4 for the following analysis of the training of a two layer subnetwork.

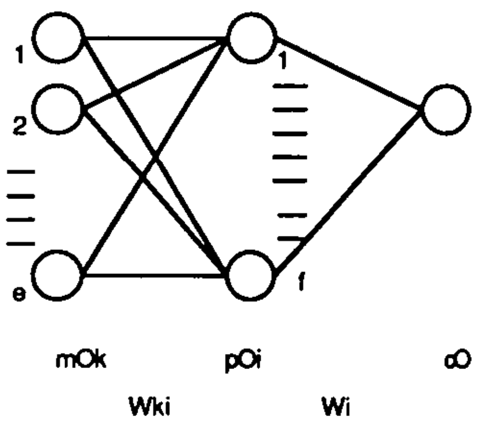


Fig 4.4 Two layer network

A two layer network consists of several different subcomponents which correspond to the situations discussed above. The inputs to the whole network are fixed Boolean inputs. The threshold functions are all sigmoid functions.

Therefore the inputs to the nodes in the hidden layer are $mO_k = \pm 1$ while the outputs of the nodes in the hidden layer are in the region $|pO_i| \leq 1$. The analysis of the misclassified points corresponds to the situation above. The update formula is;

$$\Delta mw_{ki} = -\mu\ \partial E/\partial pO_i * (df(x)/dx) \ * \ \partial x_i/\partial mw_{ki} = -\mu\ \partial E/\partial pO_i * (df(x)/dx) \ * \ mO_k,$$

$\Delta mw_{ki} = -\mu\ \partial E/\partial pO_i * (df(x)/dx) * \partial x_i/\partial mw_{ki} = -\mu\ \partial E/\partial pO_i * (df(x)/dx) * mO_k,$

$\partial E/\partial pO_i = \Sigma_j\ \partial E/\partial O_a * (df(y)/dy) * \partial y_j/\partial pO_i = \partial E/\partial O_a * (df(y)/dy) * w_i,$

$x = \Sigma mw_{ki} mO_k,$

$y = \Sigma w_i pO_i.$

An examination of the relevant formulae shows that the differences to the case discussed above, concerns the magnitude of the update based on the term $\partial E/\partial pO_i$. This shows that nodes that are weighted by a larger amount in the following layer are updated more. Corrections applied as a result of misclassifications by the neural network system can interleave one with another leading to interference on the training signal, that is the effect of $\partial E/\partial pO_i$. If a point is misclassified by the whole system but correctly classified by a given node, a form of overtraining will take place on this node. Its weight values will be updated for the particularly misclassified point, even though it is essentially correctly classified by the node.

The training of the second layer of weights is analogous to the case discussed above with $|pO_i| \leq 1$, and $|(O_d - O_a)| \leq 2$.

Multiple output nodes

Refer to the annotations of fig 4.5 for the following analysis of the training of general two layer feedforward neural networks.

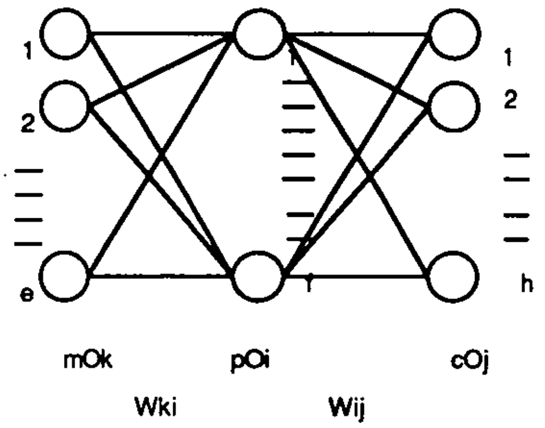


Fig 4.5 Multiple output network

The final layer of a multiple output network corresponds to the single output case and training proceeds as described in the previous section. The training of the first layer of weights must take into account the effect of the different output nodes. Each hidden node contributes to all the outputs of the network and so receives backpropagated error values from all of them. This is shown by the update equation below;

$$\Delta mw_{ki} = -\mu\ \partial E/\partial pO_i * (df(x)/dx) * \partial x_i/\partial mw_{ki} = -\mu\ \partial E/\partial pO_i * (df(x)/dx) * mO_k,$$

$$\partial E/\partial pO_i = \Sigma_j(\Sigma_l\ \partial E/\partial O_a * dfy_j/dy_j * \partial y_j/\partial pO_i) = \Sigma_j(\partial E/\partial O_a * dfy_j/dy_j * w_{ij}),$$

$$x = \Sigma mw_{ki} mO_k,$$

$$y_j = \Sigma w_{ij} pO_i.$$

This illustrates how a hidden node can receive inconsistent signals from the following layers. This can lead to destructive learning if the node or nodes model a subproblem of the input space. Constructive learning will occur if the error signals allow the nodes to converge to more accurate representations of the input data.

Intralayer communication and learning

The existence of destructive learning in the hidden layer of a neural network can be illustrated by the example shown in fig 4.6. Here two nodes isolate a given region of the input space. The effect of a single point incorrectly classified by the network is to either expand or contract the two nodes' region of influence. Each node provides a contribution to the decision of the incorrectly classified point, but taken together they effectively cancel out. Namely if +1 is provided by the nodes for the region between the nodes, then the regions outside the two nodes will have a contribution of +1 from one node and -1 from the other. A total of zero. Therefore as a whole the incorrectly classified point is not influenced by the pair of nodes and so the pair should not be affected during training. The incorrectly classified point should be modelled by another point of the network structure.

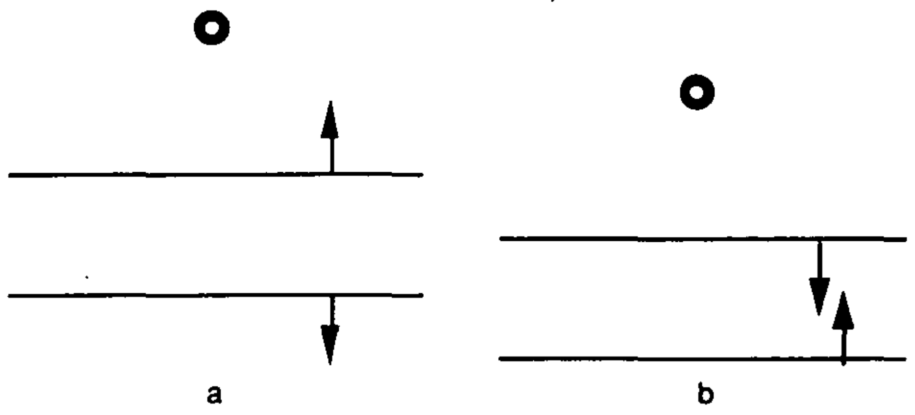


Fig 4.6 a. Expanding influence in the hidden layer, b. Contracting influence in the hidden layer

Fig 4.7 illustrates a situation where constructive learning can occur when a pair of nodes are being considered. In this case the incorrectly classified point is very near one of the nodes and so adjusting one of the nodes solves the problem.

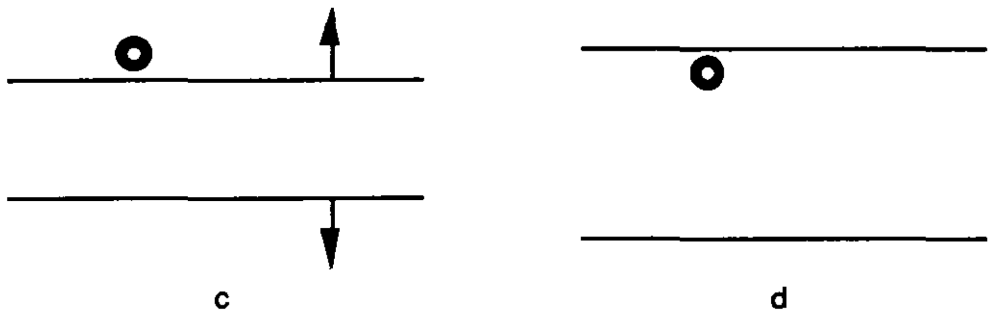


Fig 4.6 Constructive learning in hidden layer, c. Before application of training action, d. After application of training action

The overall effect of the interfering hidden nodes in general can not be predicted. Therefore this thesis proposes to structure the relationships between the nodes in the hidden layers. This formalises the interactions that occur and so allows better understanding of the network behaviour. This is further pursued in chapter five and six.

Hopfield nets

Hopfield nets are fully interconnected neural networks. The network connections are weighted and in general symmetric, that is $w_{ij} = w_{ji}$, where w_{ij} is the weight value of the connection from node i to node j. The output of a node is the weighted sum of the inputs to the node. A threshold may be applied to this value. The update of the nodes in the network may be synchronous or asynchronous. The node weights can be trained via a Hebbian learning scheme given by the formula below;

$w_{ij} = \mu \Sigma^{p}_{k=1} v^{k}_{i} v^{k}_{j}$ (for $i \neq j$), where v^{k} are the members of the training set and μ is a small constant. An iterative approach may also be adopted.

If patterns are linearly independent, a pseudo inverse approach can be adopted (Geszti '90);

$w_{ij} = (1/N)\ \Sigma^{p}_{k,l=1} v^{k}_{i} (q^{-1})_{kl} v^{l}_{j}$, where N is the number of nodes in the network,

$q_{kl} = (1/N)\ \Sigma^{N}_{i=1} v^{k}_{i} v^{l}_{j}$.

Capacity of Hopfield nets

Several results on the capacity of Hopfield nets exist. The first consideration is that of the representational power of the networks. In making capacity judgments, the limitations of the McCulloch and Pitts neurons must be considered. Each node (a McCulloch and Pitts neuron) in a Hopfield net effectively behaves as an output. Each McCulloch and Pitts neuron is incapable of modelling the parity problem and its non linear variants and so a training set that includes these properties can not be modelled by a Hopfield net. This can only be overcome with the addition of true hidden nodes. This case is discussed later in this chapter.

The pseudo inverse method of network training admits a quick limit on the capacity of the network. N-1 linearly independent patterns can be stored in a Hopfield net. This is seen from the fact that there are at most N linearly independent patterns in the N dimensional (N node) case. If there were N training patterns, the training sets would span the whole space.

This means that the training would give the weight values as $w_{ij} = 1$, $i=j$ and $w_{ij} = 0$ for $i \neq j$. (see Hertz '91, Aleksander et al '90 and Abu Mustafa '85). Therefore we can have at most N-1 linearly independent patterns.

When we have patterns that are not linearly independent, a different analysis must be made. Considering the input to each node h_i when the pattern v^k is applied to the inputs, we have the following equation;

$h^k_i = \Sigma^N_{j=1} w_{ij} v^k_j = (1/N) \Sigma^N_{j=1} \Sigma^p_{l=1} v^l_i v^l_j v^k_j$, therefore

$h^k_i = v^k_i + (1/N) \Sigma^N_{j=1} \Sigma^p_{l=1,(l \neq k)} v^l_i v^l_j v^k_j$.

Defining the crossover term c^k_i as;

$c^k_i = -v^k_i (1/N) \Sigma^N_{j=1} \Sigma^p_{l=1,(l \neq k)} v^l_i v^l_j v^k_j$, we have the condition that if c^k_i is positive and greater than one, then h^k_i will flip, that is it is an unstable node. The term c^k_i is a measure of the capacity of the net for the patterns chosen. Given a general training set, we can test c^k_i for all k and all i, to see if the patterns will be stable. If they are unstable then a different network approach must be adopted.

A general capacity measure of Hopfield nets would be useful, especially for large nets (large N) with many stored patterns (large p), since the calculation of all the c^k_i will grow exponentially in N and p. Making assumptions that N and p are large and that the training patterns are random give the capacity of the Hopfield net as $p \leq 0.138N$ (see Geszti '90 and Hertz '91). This also agrees favourably with experimental measures of the capacity, $p \leq 0.14N$ (Hertz '91 and Aleksander et al '90).

By a similar analysis the capacity of the Hopfield net, using the statistical Bolzmann execution strategy is found to be $p \leq 0.138N$.

Energy functions

A Hopfield network with a specific update strategy can be associated with an energy function. This energy function will be minimised in the execution of the network. Therefore given an energy function whose minima are the trained states, the network will converge to these trained states.

Given an energy function E defined over all the nodes h_i we have;

$E = -(1/2)\Sigma^N_{j=1} w_{ij} h_i h_j$,

$dE/dt = \partial E/\partial h_i . dh_i/dt$, where dh_i/dt is given by the update rule,

$\Delta h_i = (S_{t+\Delta t})_i - (S_t)_i$,

$(S_{t+\Delta t})_i = \text{sign}(\Sigma^N_{j=1} w_{ij} (S_t)_j)$, for the usual Hopfield update rule,

$\partial E/\partial h_i$ is defined by the weight space w_{ij}.

Optimisation

With the existence of the energy function, the Hopfield net can be used as an optimiser. If a cost function exists for a particular problem that can be defined in terms of the energy of a Hopfield net, that is all terms are linear in the nodes h_i and there are no terms of higher order than $h_i h_j$ then a network can be constructed to model the problem. When the network is executed the energy function and so the cost function will be minimised. The minimised solution may not be a global minimum and so the net must be run many times to obtain the best solution.

Training hidden nodes

Since the Hopfield model has significant capacity limitations, large networks must be used to model large training sets. The training patterns will be defined over a limited bit field and so hidden nodes must be exploited to gain the capacity to model the problem. How do we train the Hopfield net when we have these hidden nodes? Allowing the hidden nodes to settle on the most appropriate minima would be ideal. For a given input pattern the hidden node could take either value +1 or -1. One of these values will be suitable for each particular input pattern. An approach to training the net and finding suitable convergence points for the hidden nodes is via the use of a Markov model of the net. (This is discussed be Aleksander et al '90).

Markov model of the Hopfield net

The following symbols are introduced for the analysis of the Hopfield net and the states that it can represent;

Training set defined over n nodes, $D_0, D_1, \ldots D_a, \ldots, D_r$, $0 \leq a \leq r$,

$0 \leq r \leq (2^n - 1)$.

The network has N+1 nodes, N-n+1 hidden nodes, one bias node whose activity is 1. The number of possible states are 2^N,namely;

$S_0, S_1, \ldots S_p, \ldots, S_P$, $0 \leq p \leq P$, $P = (2^N - 1)$.

The activity of the i th node in state S_p is $(S_p)_i$. The activity of the i th node in training state D_a is $(D_a)_i$.

The probability of a node being active in state S_p is;

p_i = threshold$(1/(1 + \beta))$= +1 or 0, $\beta = \exp(-\Sigma^N_{j=1} w_{ij} (S_p)_j / T)$.

The probability of a node being inactive in state S_p is;

$\neg p_i$ = threshold$(\beta/(1 + \beta))$= 0 or +1.

The probability of being in a state X with the clamped environment (that is the case where each input and output node is fixed to a training value) is $P^+(X)$. The probability of being in a state X with the unclamped environment is $P^-(X)$.

$P^+(D_a)$ is 1/r since each of the trained states are equally probable,

$P^-(D_a)_{t=0} = P^+(D_a)$ can be assumed.

Markov model of unclamped state

The initial probability of being in an unclamped case is related to the probability of being in a training state;

$P^-(S_p)_{t=0} = P^-(D_a)_{t=0}/ 2^{N-n}$,where D_a is the member of the training set that corresponds to S_p over the input nodes, that is for all input nodes k, $(S_p)_k = (D_a)_k$,

$P^-(S_p)_{t=0} = 0$ if there is no D_a a member of the training set such that for all input nodes k, $(S_p)_k = (D_a)_k$.

The progression of this system in time is given by;

$P^-(S_p)_t = \Sigma^P{}_{b=0} P^-(S_b)_{t-1}\ p(b,p)$, where p(b,p) is the probability of passing from state S_b to S_p in one bit change,

$p(b,p) = 0$, for more than one bit change,

$p(b,p) = p((S_b)_i)/N$, for $(S_b)_i \neq (S_p)_i$ and $(S_b)_j = (S_p)_j$,for $j \neq i$,

$p(b,p) = \neg p((S_b)_i)$, for $S_b = S_p$, and

$p((S_b)_i) = p_i$, if $(S_b)_i = -1$, or $p((S_b)_i) = \neg p_i$, if $(S_b)_i = 1$.

These equations allow the converged state of the network to be calculated. The probability of the trained states occurring can be calculated from the relevant unclamped states;

$P^-(D_a) = \Sigma^P{}_{b=0} P^-(S_b) r(a,b)$, where r(a,b) is a relevance measure such that,

$r(a,b) = 1$ if $(S_b)_k = (D_a)_k$ over the input nodes k,

$r(a,b) = 0$ otherwise.

Markov model of the clamped state

The initial probability of being in a clamped state is evenly distributed over the hidden nodes;

$P^+(S_p)_{t=0} = P^+(D_a) / 2^{N-n}$, where D_a is the member of the training set that corresponds to S_p over the input nodes, that is for all input nodes k, $(S_p)_k = (D_a)_k$,

$P^+(S_p)_{t=0} = 0$ if there is no D_a a member of the training set such that for all input nodes k, $(S_p)_k = (D_a)_k$.

When a network is running in a clamped mode only the unclamped nodes are allowed to change. This means that at each update the network states perturbs and then these states are clamped. This situation is modelled by the equations;

$P^+(S_p)^*_t = \Sigma^P_{b=0} P^+(S_b)_{t-1}\ p(b,p)$, see previous definition of $p(b,p)$,

$P^+(S_p)_t = \Sigma^P_{b=0} P^+(S_b)^*_t\ d(b,p)$, where $d(b,p)$ is a relevancy measure when the network is clamped,

$d(b,p) = 1$, if $(S_p)_j = (S_b)_j$, for all j not input nodes,

$d(b,p) = 0$, if there exists j such that $(S_p)_j \neq (S_b)_j$, where j is not an input node.

After letting the Markov model converge the probability of the trained states occurring can be calculated from the relevant clamped states;

$P^+(D_a) = \Sigma^P_{b=0} P^+(S_b) r(a,b)$, where $r(a,b)$ is defined above.

Training

The local update rule for the weights is;

$\Delta w_{ij} = -\Delta G.T/(\ p^+_{ij} - p^-_{ij})$, where G is the information function $G = \Sigma_a P^+(S_a) \ln[P^+(S_a)/P^-(S_a)]$. Since $G = 0$ is the minimum, we can set $\Delta G = G$, where p^+_{ij} is the probability that unit i and j are both active in the clamped environment and p^-_{ij} the probability that they will both be active in the unclamped environment. The values of p^+_{ij} and p^-_{ij} can be calculated from the Markov models above via the formulae;

$p^+_{ij} = \Sigma^P_{b=0} P^+(S_b) f(i,j,S_b)$,

$p^-_{ij} = \Sigma^P_{b=0} P^-(S_b) f(i,j,S_b)$, where $f(i,j,S_b)$ is a measure of the relevance of states S_b, such that,

$f(i,j,S_b) = 1$, if $(S_b)_i = (S_b)_j = 1$,

$f(i,j,S_b) = 0$, otherwise.

The network can be trained by this mechanism but will require a great deal of computation. A similar performance can be obtained by judiciously selecting the hidden node values and training the Hopfield net as if there were no hidden nodes and giving the hidden nodes these specified values during training.

Input pattern coding

In order that a Hopfield net can model the training set, each training pattern must be defined over all the nodes, including the hidden nodes. This means that an almost arbitrary coding of the training pattern over the input nodes must be applied over all the nodes. This network must be capable of modelling the training set, so several criterion must be satisfied. Each node in the network is a McCulloch and Pitts neuron and so is only capable of modelling thresholded linear transformations. Therefore the output node must be a linear transformation of the input and hidden nodes for the network to be capable of modelling the training set. This must also be true of the hidden nodes in the network. These considerations provide a constraint on the values the hidden nodes can take over the whole training set.

If a large network exists, random patterns over the hidden nodes may solve the problem. However a more structured approach must be adopted to ensure suitable performance.

The training algorithm for Hopfield nets make use of the correlation factor between the two nodes concerned to specify the weight;

$w_{ij} = \mu \, \Sigma^{p}_{k=1} v^{k}_{i} v^{k}_{j}$ (for $i \neq j$). This value depends on the hamming distance between the bit vectors v_i and v_j . Namely;

$\Sigma^{p}_{k=1} v^{k}_{i} v^{k}_{j} = p - 2$*hamming_distance(v_i, v_j). From this equation it can be seen that the correlation of two nodes can be increased by decreasing the hamming distance between the nodes. A non linearity between input and output nodes is often associated with a low correlation between these nodes. Providing hidden nodes with a small hamming distance from both the input and output nodes provides the possibility of modelling the training set with a Hopfield net. The hidden nodes provide a high correlation path between input and output.

A necessary and sufficient criterion for linearity of the hidden and output nodes based on these correlation values can not be found. This is illustrated by the examples in tables 4.1 a, b, c, that provide identical correlation values for the respective nodes but are linear and non linear transformations respectively.

pattern \ node	0	1	2
0	-1	-1	1
1	-1	1	-1
2	1	-1	-1
3	1	1	-1

a

pattern \ node	0	1	2
0	-1	-1	1
1	-1	1	1
2	1	-1	-1
3	1	1	-1

b

Table 4.1 a. & b. Linear transformations

pattern \ node	0	1	2
0	-1	-1	1
1	-1	1	-1
2	1	-1	-1
3	1	1	1

Table 4.1 c. Non linear transformation

Therefore the correlation values of the Hebbian learning technique do not guarantee correct representation unless we have a linear relationship between the input and output nodes. It is the linearity of the transformation that is the significant factor and not the correlation factors that determine whether a network can be trained. If a linear transformation exists then the Hebbian learning scheme is valid otherwise new network models with different hidden node values must be investigated.

Neural network reliability

A neural network can be trained on a set of data. Its ability to model this training set and the ability to generalise to points not in the training set depends on the reliability and predictability of the neural model. To formalise these ideas a mathematical definition of reliability is given.

Reliability of network output = Probability that output is correct.

Given a network with a single output node and taking the simplified model of network training, that is the network converges to model the training set perfectly but is unable to predict the output over the other input values, we can give the following reliability measures;

Probability(correct output|training point) = 1,

Probability(correct output|not training point) = 1/2,

n = number of input nodes, N = number of training points,

then;

Reliability = Probability(correct output) = $(N + 2^n) / 2^{n+1}$.

Representational reliability

Representational reliability of a neural model is dependent on the uncertainty in the transformation that is being modelled. Since there are a large number of possible transformations for a given number of inputs, the probability of correctly modelling a transformation decreases dramatically as the training set is reduced.

Representational reliability = Probability that transformation is correctly modelled

Therefore given a perfectly modelled training set;

Probability(correct output|training point) = 1,

Probability(correct output|not training point) = 1/2,

n = number of input nodes, N = number of training points,

$N^* = 2^n - N$ = number of input points not in training set,

then;

Representational reliability = Probability(correct model) = $(1/2)^{N^*}$.

Summary

The various training algorithms that exist for creating neural models of data have been examined. These include those for feedforward systems, those based on backpropagation and those for recurrent networks, such as Hopfield and Bolzmann nets, which use variations on the Hebbian training rule.

The local node level effect of the backpropagation training algorithm was examined and the types of internode interference that can occur discussed. Two reliability measures of the trained networks were introduced as a formal technique for examining the effect of training set size on the reliability and predictability of neural models. These measures will be used to analyse the effect of node parallelisation and similar structural techniques on neural network reliability (see chapter five and six).

Chapter 5. Parallelisation of Nodes in the Hidden Layer

Outline of chapter

The use of node parallelisation in neural systems is introduced. Parallelisation in the hidden layer is introduced as a mechanism of introducing interdependencies between the nodes. It is seen that this is one of the basic mechanisms for providing the training algorithms with knowledge of the node representations in the hidden layer. It is shown that sandwich parallelisation is natural in binary feedforward networks, and are indeed essential for minimal representations of some transformations. A training algorithm is presented making use of parallel nodes (ghost nodes). Sandwich parallelisation and polygonal segmentation are examined as techniques for structuring the nodes in the hidden layers of feedforward neural networks.

Introduction

Throughout this chapter we will be discussing bipolar feedforward network systems. As previously demonstrated (Hinde '90), these networks only require one hidden layer of nodes. Further work (chapter seven), addresses the number of hidden nodes required. The network is fully interconnected between the layers and can represent any Boolean transformation, given the number of input and output units.

In all the studies to date no structural dependencies have been applied to the hidden layer of a neural network (Hinton '89, Rumelhart et al '86). This was done so that there would be no initial constraints on the network representations. The training was allowed to proceed freely in order that it may converge to an optimal solution. Any extra constraints of internal structure were neglected. This may have been because authors felt that they may inhibit the training process, although none explicitly mention this problem. This lack of structure may be deemed appropriate for biological reasons, Wasserman ('72) however, shows that biological neural linkages are predetermined to a large extent and that the topology of the brain is highly determined at birth.

The main disadvantage of standard training algorithms, that is backpropagation, is the way they treat each hidden node identically and in isolation. When the weight space of a hidden node is updated, only the effect of that node on the error function is taken into account. The way that the hidden node interacts with the rest of the hidden layer in order to model the transformation is not taken into account with this approach.

In the case when a hidden node correctly models a subspace of the training set over which it makes the most significant contribution, then it is better to leave it unperturbed rather than spuriously adjusting it for short term gain in error minimisation. It would be better to perturb another node in the hidden layer to ultimately produce a better model for the incorrectly classified training examples. This will become more apparent when we examine sandwich nodes and their behaviour under the learning algorithm.

Short term error minimisation often leads to the undesirable property that existing neural representations that model the transformation well, are knocked out to produce non optimal locally minimal representations. Introducing some structure and dependencies into the hidden layer ensures that adjustments to each hidden node can act in concert with the rest of the representation. The simplest dependency relation between nodes is that of parallelism which is discussed below.

Nodes are considered to be parallel if they have identical weight vectors (to a scalar factor) defined over the same inputs, with possibly distinct bias weights. The ideas discussed in this chapter are more simple, essentially that node parallelisation is implemented via duplications in weights space, namely one or more nodes shadow or ghost the reference node, their weights are just a duplicate of the reference node or its simple negation.

Parallel or as introduced in Messom ('92) ghost nodes allow for general duplication of hidden nodes (their structure are discussed later), which can then be used as parallel nodes that output to a single output node, providing a dependency between the hidden nodes of a particular neural net.

Structure in the hidden layer

Two types of dependencies are investigated, both are closely related.

i. Parallel hidden nodes.

ii. Sandwich hidden nodes.

i. Parallel hidden nodes;

are defined by the criterion that the bias weights are independent while all the other weights are pairwise identical. That is, given two nodes, they are Parallel if the weight vectors are identical except for the bias factors. (See fig 5.1a).

ii. Sandwich hidden nodes;

are defined by the criterion that the bias weights are independent while the other weights are pairwise additive inverse. That is, given two nodes, they form a sandwich if the weight vectors are additive inverses except for the bias factors. (See fig 5.1b).

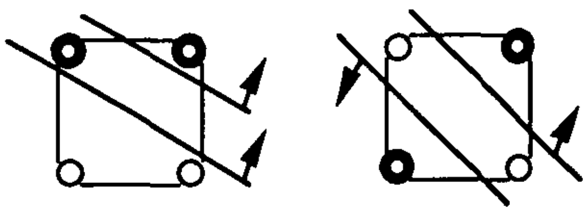


Fig 5.1 a. Parallel nodes in two input space, b. Sandwich nodes in two input space

Sandwich nodes are a pair of close opposite facing nodes that isolate a small subspace of the input region. The advantage of using sandwich nodes over arbitrary single nodes is that the sandwich node will isolate a given region, which it will contribute a positive or negative decision, while not contribute significantly to the outer regions. That is the sandwich node behaves as an atomic declaration of truth over the input space. This means that the overall definition of the input space can be constructed via a number of atomic declarations.

Introducing the dependencies above offer several advantages. The first advantage is one of representation. Introducing dependencies in the hidden layer, reduces the variety of representations of the given transformation. This gives more structure to the network representation. The networks can then be reasoned about, since their structure will be transformation dependent.

The second advantage of the interdependent nodes in the hidden layer is the natural improvement in computational performance that it offers. The weights being equal means that the weight component of the input of a node need only be calculated once, rather than every time each node is passed. The ghost system offers this property of a reduced computational load.

Ghost nodes

Ghost nodes are nodes that are dependent on at least one other node in the hidden layer. There are two basic dependencies that exist corresponding to the parallel and sandwich cases.

i. the ghost nodes are parallel, this means that they have identical weight spaces, that is they share a weight space vector, although they have independent bias weights.

ii. the ghost nodes are antiparallel, this means that the two nodes have weight spaces that are additive inverse, although they have independent bias weights. These antiparallel ghost nodes also share a weight space vector although one node must apply a negate the vector before making use of the weights.

Training

The training of ghost nodes proceeds in a similar manner to standard backpropagation. Each ghost node is updated by the backpropagation algorithm in a manner proportional to its contribution to the error function. However since the ghost nodes shadow each other any update on one ghost node is also applied to the weights of the corresponding ghost nodes, that is the shared weight vector is updated each time one of the ghost nodes are trained. The bias weights are the only weights that are not updated when corresponding ghost nodes are updated.

Segmentation of the input space

Two antiparallel planes that are distinct, that is not coplanar isolate a segment of the input space. The sandwich so formed provides an output of say +1 for the region between the planes, while for the region outside the sandwich there is an effective output of zero, since the individual planes contribute +1 and -1 which effectively cancel. A neural network consisting of a single pair of nodes forming a sandwich effectively segments the input space into three regions. That inside the sandwich providing an output of +1 say, while for the two regions outside the sandwich provides the output -1. (The -1 is achieved in the region where the sandwich node effectively makes no contribution by providing a bias of -1 on the output node of the network).

The single sandwich can be implemented by a pair of ghost nodes that are antiparallel. Similarly we can segment the input space into any number of parallel regions with an array of antiparallel planes, (see fig 5.2). This is achieved by interleaving two sets of ghosted nodes that are all parallel but mutually antiparallel between the two sets.

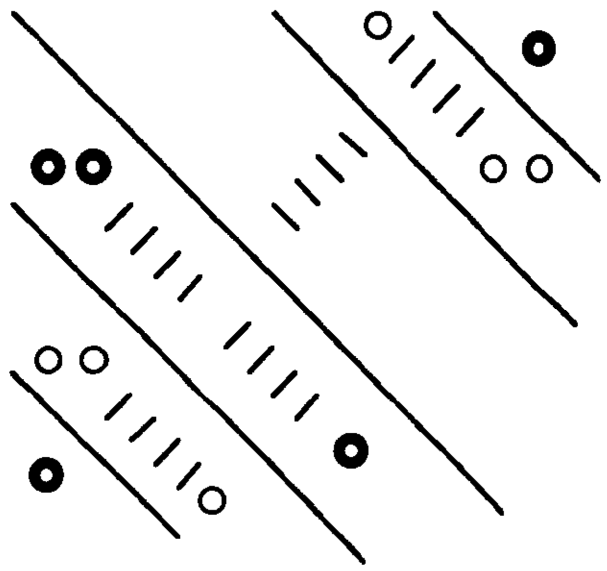


Fig 5.2 Segmentation of the input space

In this way, a region between two antiparallel planes will provide an output of say, +1, while its neighbouring region, contained by the neighbouring pair of antiparallel planes will provide the output -1. And carrying through this analysis for the whole input space, we have parallel regions where the output value is either +1 or -1. Therefore we can segment any input space into arbitrary bands of parallel regions with a paired set of interleaved ghost nodes.

Utility of ghosted segmentation

The utility of the parallel segmentation of the input space using ghosted arrays of antiparallel nodes comes to light when we consider general Boolean transformations from multiple inputs to a single output. Essentially any transformation can be implemented as a parallel segmentation of an input space. Each parallel segmentation yields a unique transformation, but of course each transformation can be modelled by many parallel segmentations. We can see the truth of these statements by following the analysis below.

i. Each parallel segmentation yields a unique transformation;
This follows directly from the fact that we can implement a neural representation of the parallel segmentation of the input space, which can in turn provide output values for specific Boolean input values. This in turn defines a transformation which is that yielded by the original parallel segmentation of the input space.

ii. Each transformation can be modelled by many parallel segmentations;
This is less obvious but can be proved by construction. Consider a point in the Boolean input space, say a. There exists a hyperplane that goes through a but does not intersect the hypercube that is the whole input space except for the point a, which also is not parallel to any plane that goes through any two points in the Boolean input space. There is a plane parallel to this one that goes through the point **not**(a), but does not intersect the hypercube that is the whole input space except for the point **not**(a).

We can continuously transform the first plane going through point a until we reach the second plane going through point **not**(a). As we do this, the plane passes through every point in the Boolean input space, so inducing an ordering on the Boolean input space, beginning at point a and ending at point **not**(a). This ordering is a strict ordering by the criterion that the original plane is not parallel to any plane going through any two points in the Boolean input space.

Any change in output value as we examine two neighbouring input values in this ordering of input points, can be implemented in a network representation by the addition of an appropriate ghost node parallel or antiparallel to the original plane through point a , which passes through the point that bisects the line between the two input points in question. Examining the whole ordering yields a ghosted network of interleaved parallel and antiparallel nodes. This ghosted network yields a parallel segmentation of the input space.

Choosing another point say b and another suitable plane and carrying through the analysis would have yielded another, different segmentation of the input space. Therefore any transformation can be modelled by many parallel segmentations of the input space.

Representing transformations via ghosted parallel segments

The analysis above proves that any transformation can be modelled by an array of ghosted parallel and antiparallel nodes. Finding this ghosted network that is suitably minimal provides the major difficulty. Constructing such a network following the analysis above does not guarantee minimality and may require networks with a large number of hidden nodes. This problem is overcome by allowing the network to be trained by the ghosted backpropagation training scheme. (See appendix C).

The parity transformation for n inputs can be modelled by n hidden nodes. This is the minimal representation for the n dimensional parity problem and in fact can also be modelled by a suitably ghosted system using just n ghost nodes. This ghosted representation can be constructed or discovered by training. It should be noted that the convergence algorithm will often get stranded in local minima and so must be carefully monitored.

Models of the parity problem using ghost nodes

Parity in two dimensions as discussed in chapter three is the exclusive or problem. This could be modelled by two parallel nodes. The network structure is shown in fig 5.3, while the weight matrix that define the hidden layer are shown in fig 5.4a. The weights shown are in the form (bias, weight_1, weight_2, ..,weight_n).

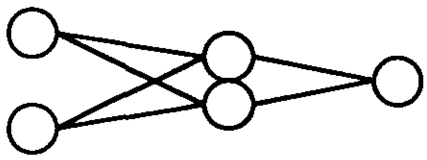


Fig 5.3 Ghosted neural network model of the parity problem

$$\begin{pmatrix} 1, -1, -1 \\ 1, 1, 1 \end{pmatrix} \qquad \begin{pmatrix} -2, 1, 1, 1 \\ 0, -1, -1, -1 \\ 2, 1, 1, 1 \end{pmatrix} \qquad \begin{pmatrix} 3,-1,-1,-1,-1 \\ -1, 1, 1, 1, 1 \\ -1,-1,-1,-1,-1 \\ 3, 1, 1, 1, 1 \end{pmatrix}$$

a b c

Fig 5.4 Weight matrices for the parity problem in a. two, b. three, and c. four dimensions

Fig 5.4b and fig 5.4c show the matrices for the parity problem in three and four dimensions. Parity in general dimensions follows the same pattern. If a given input vector has the output +1, then the same vector perturbed by just one bit will give the output value -1. We can construct the general matrix of weights for parity by starting with the point (-1,-1,...,-1) and traversing the input space to the point (1,1,...,1). We segment the input space into parallel regions that provide the outputs +1, and -1 respectively. These regions are formed by the planar boundaries that have the structure, just one input has value +1, no more than n-2 inputs have value -1 etc. Matrix representation is distinct for odd or even number of input nodes due to the structure of the problem. An odd number of input nodes that all have the value +1, will require the output +1, while an even number of input nodes that all have the value +1, will require the output -1. Fig 5.5 a & b show the weight matrix of the hidden nodes of the neural network models of parity.

$$\begin{pmatrix} -(n-1), 1,\ldots,\ldots,1 \\ n-3,-1,\ldots,\ldots,-1 \\ -(n-5), 1,\ldots,\ldots,1 \\ \vdots \qquad \vdots \\ n-5, 1,\ldots,\ldots,1 \\ -(n-3),-1,\ldots,\ldots,-1 \\ n-1, 1,\ldots,\ldots,1 \end{pmatrix} \begin{pmatrix} n-1,-1,\ldots,\ldots,-1 \\ -(n-3), 1,\ldots,\ldots,1 \\ n-5,-1,\ldots,\ldots,-1 \\ \vdots \qquad \vdots \\ n-5, 1,\ldots,\ldots,1 \\ -(n-3),-1,\ldots,\ldots,-1 \\ n-1, 1,\ldots,\ldots,1 \end{pmatrix}$$

a b

Fig 5.5 Matrix of weights for the parity problem with a. Odd number of input nodes, b. Even number of input nodes

Ghosted nodes for real valued inputs

We can apply the ghosting techniques to certain classes of transformations from real valued input spaces to Boolean output spaces.

Transformations from input spaces whose areas of interest are, or can be approximated by, parallel segments or intersections of parallel segments are ideal for implementation via the ghosting procedure. See fig 5.6 & 5.7 for examples of segmentation of the real line and plane.

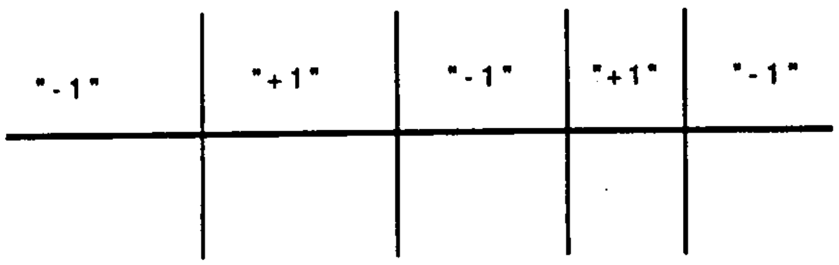


Fig 5.6 Segmentation of the real line

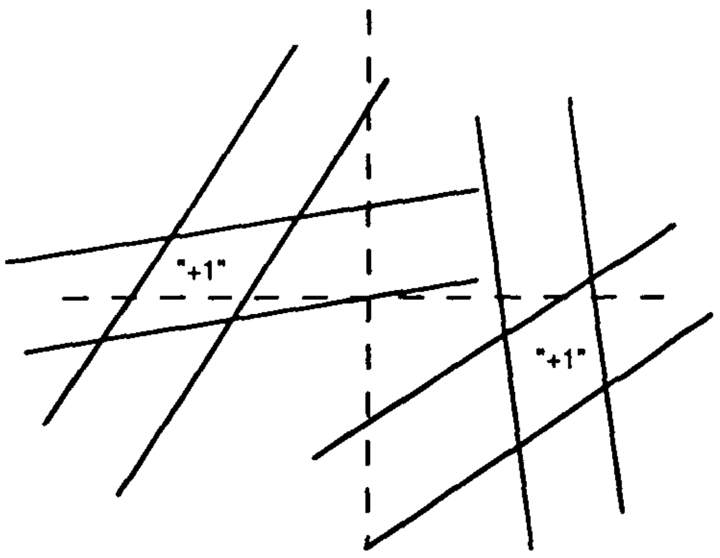


Fig 5.7 Segmentation of the real plane, forming two polygonal regions

Encapsulated ghost nodes

Using ghost nodes in the neural network system ensures that the structure of the hidden nodes is well defined. This does not mean that the backpropagation algorithm proceeds without error interference and destructive learning (as discussed in chapter four). A method that can be employed to reduce internode error interference is to add an extra layer of nodes that encapsulate the sandwich node function. The network structure is illustrated in fig 5.8 which shows the sandwich nodes which encapsulate the function of two planar nodes. With this network structure the error effects of incorrectly classified points are not back propagated to sandwich units that do not significantly contribute to the error.

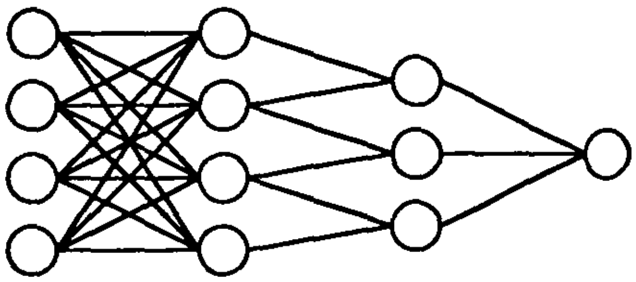


Fig 5.8 Encapsulated sandwich node network structure

If a training point is significantly away from the pair of planar nodes that make up the sandwich, the contribution of each is approximately equal but opposite in sign effectively cancelling each other, so stopping the error backpropagation. If the training point is close to at least one of the planar nodes, then significantly different contributions will be made by each node so providing a contribution to the error from the sandwich pair. In this case error backpropagation and weight update occurs.

Polygonal segmentation

A sandwich node is constructed from two planar nodes. Similarly more complex encapsulated regions can be constructed by employing more nodes. Three nodes form a triangular segmentation region, this is illustrated by the fig 5.9. The network structure that can support this encapsulated polygonal segmentation is shown in fig 5.10. More planar nodes can be employed to form encapsulated polygonal and polyhedral segmentation units.

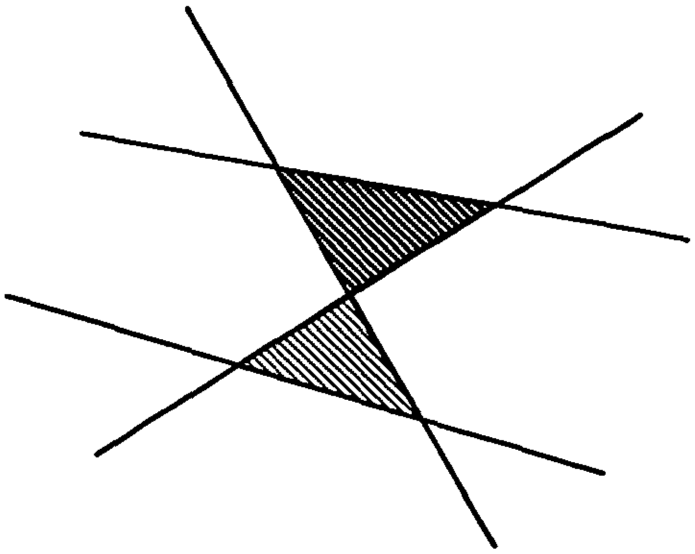


Fig 5.9 Segmentation of a two dimensional space into triangular regions

The encapsulated polygonal segmentation units have the same advantage as the encapsulated sandwich nodes of non interfering error backpropagation. The encapsulation of node representations is discussed in greater detail in chapter six.

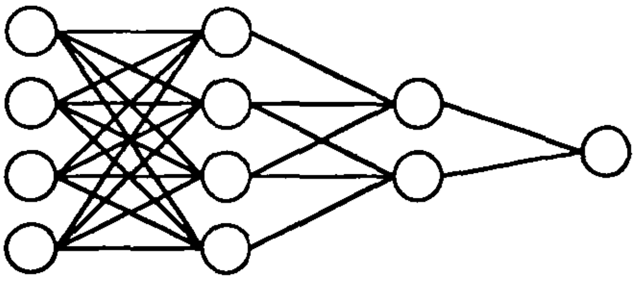


Fig 5.10 Network structure employing encapsulated triangular units

Summary

This chapter has introduced node parallelisation as a mechanism for structuring the hidden layers of a neural network. Its utility in modelling several transformations, particularly parity, has been demonstrated. Parallel nodes as a scheme for segmenting real valued input spaces have been investigated and shown to be a suitable mechanism for constructing linear quantisers.

Finally, encapsulated sandwich and polygonal segmentation nodes were introduced as a mechanism for solving the intralayer error interference problem that causes destructive learning under backpropagation training.

Chapter 6. Making use of Knowledge in Neural Nets

Outline of chapter

The major problem associated with neural network systems is the lack of structure to the representation. Each transformation is fully represented by a large network structure often with a large degree of distributed activity. This property restricts the ease with which the behaviour of the network can be predicted. It also is almost impossible to combine properties of different networks to produce a more reliable network structure.

Neural network representations lack a knowledge structure with which they can be successfully reasoned about. In this chapter several new structures are introduced which go some way towards solving this problem.

The parallel sandwich node is introduced as the atomic element of a monotonic neural system. The concept is extended to general sandwich nodes which allow a knowledge representational scheme to be built. This scheme allows the neural network subsymbolic reasoning processes to be formalised and encapsulated. Finally the reliability of sandwich node systems are investigated, showing that even when they are trained over reduced data sets the neural network behaviour is reliable and predictable.

Knowledge representations of neural networks

The simplest knowledge structure is the neural network node itself. These can be combined in a predetermined manner to produce more complex knowledge structures which can be manipulated and interpreted in a well defined symbolic manner. This introduces the possibility of providing a neural network environment with properties more familiar in knowledge engineering, the atomicity of knowledge and the monotonicity of information. Building upon these basic structures, well defined nonmonotonicity can be introduced as information is updated and manipulated.

Neural network behaviour is hard to predict. Neural networks have been modelled as systems of Boolean transformations and as rule systems but neither offers a high level model of the subsymbolic behaviour of the neural networks.

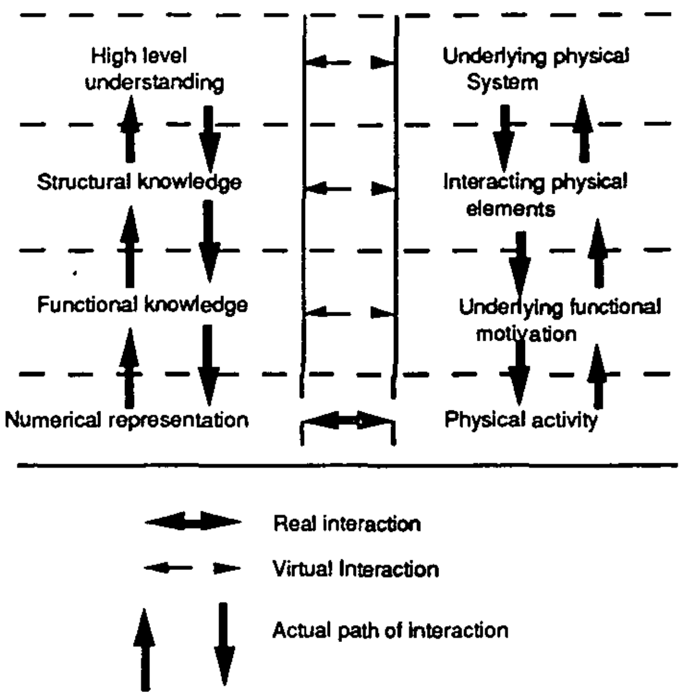


Fig 6.1 Interaction between models of neural networks and actual neural network systems

The need for a high level model of neural systems can be seen with the system interaction model in fig 6.1. This model draws on the three level human-computer interface model of Clarke ('86), which discusses the different levels of interaction between user and machine. In fig 6.1 we examine the various levels of interaction between a neural implementation of a physical system and the human model of the neural network.

At the basic bit level we have the one to one mapping between the neural network and its mental model. Each local process in the neural network is understood and any particular interacting elements of the network process can be understood. A higher level understanding of the network depends on the functional model of the network. In the neural network side of the model this may be the functional declaration of the neural network processes. The mental model would be an understanding of the functional processes.

As demonstrated by Clarke, the path of interaction between the two sides of the model is via the base level of the model, that is via the neural network implementation, so even if the underlying functional model is well known, it is manifested via the neural network behaviour. We can develop a higher level model of the neural network, a Boolean model or a rule based model, but even these are unreliable if they are derived from the neural network behaviour over a limited number of input cases. If these models corresponded to the interacting physical systems that produced the network behaviour itself then we could construct a reliable predictive model of the neural network. This is not the case with the training schemes used at present. A general prescriptive model of the neural network based on well defined knowledge elements within a neural network that interact in a well structured manner will provide the basis for a high level model of the subsymbolic processes within the neural network. The high level model of the neural network will begin to allow a high level model of the relevant physical system to be derived.

Producing models of general physical systems will require sophisticated knowledge elements within the neural network. Particularly simple knowledge elements such as sandwich nodes and parallel nodes are discussed in this chapter in the hope that future work will build on these structures to provide suitable elements to model general physical systems (see chapter nine).

Knowledge elements and subsymbolic reasoning

The processing in neural networks have been described as subsymbolic. All the transformations from input to output take place at the bit level. This is the case as only simple -1 and +1 signals pass between the neuronal processing elements. The processing elements at each node treats all messages from different units identically and so the bit level activity can not be viewed globally as a high level message passing paradigm.

Neural networks admit a subsymbolic reasoning model of their behaviour as follows. The network is trained on sample data points, the training data which will be correctly modelled by the network after optimal convergence of the training algorithm. This training set can be viewed as the knowledge base of the neural network. The neural network execution admits an inference engine on the knowledge base. Every input pattern produces a specific output from the network and so this output can be viewed as having been achieved by a process of subsymbolic reasoning. If the input point was part of the training set, then the stored data point will yield the output that is stored in the knowledge base of the network. If a novel input pattern is presented, a form of subsymbolic inference on the knowledge base takes place yielding an inferred output.

The great disadvantage about the subsymbolic reasoning processes is that they are generally ill defined. Different network structures that model the same training data will apply different subsymbolic processes yielding often distinct outputs. This is extremely undesirable if subsymblic reasoning is to be reliable and in some sense predictable.

Symbolic reasoning systems have the property that every conclusion reached can be explained simply by the presentation of the subset of the rule base used to reach the conclusion. Neural networks in general have a great deal of distributed activity that can not explain a conclusion without quoting the structure of the whole network. The existence of diverse subsymbolic processing schemes for distinct networks means that a general explanation can not be implemented either. Each unique network can have its own explanation scheme based on the execution strategy of the specific network.

Knowledge elements within neural networks go some way towards formalising subsymbolic processing within neural networks to the point that they can be relied upon, understood and explained.

Formalising subsymbolic systems

Subsymbolic reasoning in neural networks is extremely network dependent. Introducing the concept of atomic knowledge elements into neural systems allows us to develop a general framework in which subsymbolic reasoning can be discussed. A subsymbolic paradigm can be modelled as a formal symbolic system if we can provide atomic formulae and the combinations and transformations that can be applied. The subsymbolic paradigm must therefore have;

a. atomic knowledge elements; the atomic formulae of the subsymbolic system.

b. well defined interconnection properties; the transformations that can be applied to the atomic knowledge elements.

In a neural network environment the atomic knowledge elements are nodes, sandwich nodes, subnetwork structures or other more complex atomic elements. It should be noted that sandwich nodes are not atomic in the sense that they are constructed from two nodes, but are indeed atomic knowledge elements since they specify atomically whether an input pattern is within a region or not and there is no complex interaction within the sandwich node.

The interconnection properties relate the possible firing patterns of the atomic knowledge nodes and how they can be connected to form the output.

Given this formal structure of the knowledge neural networks, training can proceed in two ways.

i. Specified; the network is constructed on the basis of the properties of the knowledge atoms and their connection properties. The behaviour of this network will be perfectly predictable since it has been specified.

ii. Learned; the network is presented with sample data points and trained, subject to the constraints of using the atomic elements and the specified interconnection patterns allowed. These constraints may limit system identification but if knowledge about the system exists, the network can be explicitly structured with suitable knowledge elements to improve the automated system identification. On convergence the subsymbolic reasoning properties can be predicted by examining the knowledge atoms employed by the network and the specific connection patterns employed by the converged network.

The execution of the network can be viewed as a subsymbolic reasoning process that can provide explanation of the conclusion reached. An input data point is presented to the network. An output is computed which is explained by the following sequence.

i. Disclose the atomic knowledge elements that fired.

ii. Disclose the interaction pattern or connection pattern that was employed to reach the conclusion.

iii. Disclose the training patterns that have similar classifications and corresponding connection and firing patterns.

This explanation sequence recognises an input pattern that is one of the training data and gives its relevant constructed knowledge class. Given an input pattern that is not a training point, it will identify the point's relevant class, explain the structure of that class in terms of the fired atomic elements and also present examples of the training set that have the same class. Such a neural network system can be fully understood and so prove to be reliable.

Interpretation of neural networks

A feed forward network system can be naturally interpreted as a functional transformation. Therefore, there always exists an interpretation of the network at this lowest level. For large systems, this form of interpretation is cumbersome and incomprehensible. This means that a more structured approach must be adopted.

The interpretation of a network representing a Boolean transformation is well documented. The transformation can be seen to be implemented by a set of logical rules, this is discussed by Hinde ('90) and in chapter three. However these basic rule systems are not easy to interpret, each nodes contribution being viewed in a possibly verbose logical form rather than an arithmetic functional form.

The logical interpretation of networks can be extended to the analysis of parallel ghost nodes. The sandwich nodes are an aid to interpreting ghost node network structures. The structure of these nodes should be maintained, even when viewing them as logical transformations since these structural units in the networks provide atomic non interfering encapsulated representations about which we can reason. Reasoning about the behaviour of the net is simplified with these sandwich and ghost concepts.

Sandwich nodes

A sandwich node is a pair of close opposite facing nodes. Formally this means that a single sandwich node is a pair of weight vectors, which are mutually inverse and the biases are close. The condition that the biases are close is satisfied if there are no other nodes in the network parallel to the sandwich node which have scaled biases larger than one bias but smaller than the other. This means that there are no other nodes in the network that are between the two nodes in the sandwich.

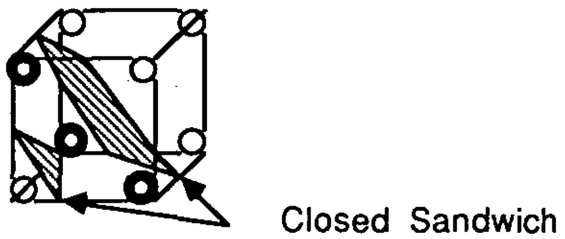


Fig 6.2 a. A sandwich node in three space

Such a sandwich can be viewed as an isolating element in network. Over the region the sandwich isolates, it is the only contributing element. Everything that is stated about this region is only provided by the sandwich, (see fig 6.2a). Similarly we can consider an open sandwich.

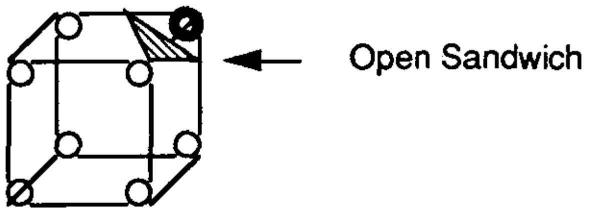


Fig 6.2 b. An open sandwich node in three space

When a single node isolates a region such that all other nodes are nil-separating hyperplanes of that space, then essentially that single node sandwiches off that region. It is the only node that contributes to that region, adding just a bias weight to the other side of the sandwich. (See fig 6.2b).

Pairs of antiparallel ghost nodes (see chapter five) can be considered to be single conceptual structures called sandwiches. Individual nodes that are not ghosted can be considered to be open or half sandwiches, if they are nil contributing in each half space of the other hidden nodes. A node is nil contributing in a subspace if its contribution in this region is a fixed value.

Formalised subsymbolic reasoning with sandwich nodes

General sandwich nodes have the following formal structure which defines an atomic knowledge element. The definition of a sandwich node can be extended to that of a node formed by two nodes that do not intersect within the input space, that is they de not have to be parallel. A single node can be classed as an open sandwich node.

The interconnection constraints of a sandwich system are defined by the constraint that the sandwich nodes in each layer must not intersect in the input space and must not be nested.

Due to these constraints only a single closed sandwich node or a coherent set of open sandwich nodes will fire in each layer. This is the case since each sandwich node isolates a convex subspace of input space and no sandwich nodes overlap, therefore if a sandwich node fires, only one will fire. A set of open sandwiches can coherently isolate a convex subspace of the input space, since they do not intersect each other or any sandwich node, therefore if a set of coherent nodes fire, then no other coherent set of open nodes will fire.

These criterion ensure that the input space is structured into well defined convex spaces in which a specified output can be defined. The output layer then consists of selecting suitable subsets of the convex spaces such that the required output is given. These properties can be illustrated by considering several simple examples with the relevant properties.

Examples

To simplify the discussion and the diagrams employed, the following examples are from an input space of two dimensional real values with an output space of Boolean values. The analysis is valid for higher dimensions and subsumes Boolean input spaces.

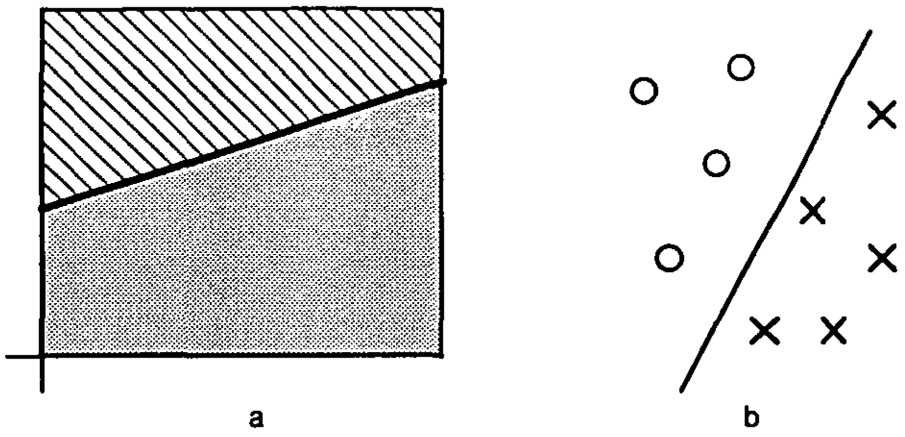


Fig 6.3 a. A decision region of an open sandwich node, b. Linear separation by an open sandwich node

A single open sandwich can isolate any linearly separable set of samples. One side of the node gives an output of +1, the other -1. See fig 6.3 a & b.

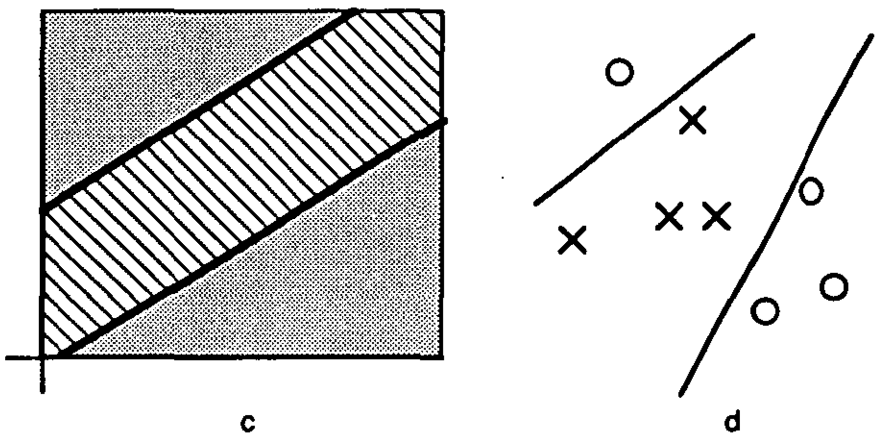


Fig 6.3 c. Decision region of a closed sandwich node, d. Segmentation by a closed sandwich node

A single sandwich node isolates a space with "exclusive or" properties. These sample points are not linearly separable. The sandwich node gives an output of +1 within the sandwich and -1 outside of the sandwich. See fig 6.3 c & d.

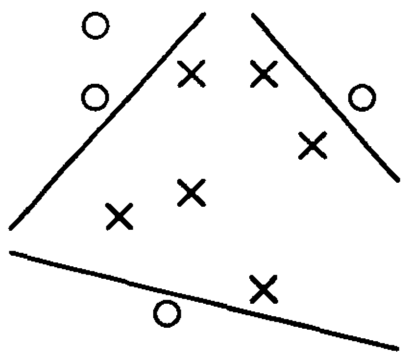


Fig 6.4 Coherent segmentation with a set of open sandwich nodes

A set of coherent sandwich nodes isolate convex region of the input space (see fig 6.4 for an illustrated example). Each node provides an output of +1/3 on the positive side of the node and an output of -5/3 on the negative side of the nodes. These node values give a coherent combination of the outputs to provides value of +1 for the convex region within the open sandwich nodes and a value of -1 in the region outside of the set of open nodes.

Interacting sandwich nodes

For general sets of data that can not be isolated by a single convex region, a set of interacting sandwich nodes must be employed. In fig, each sandwich node can provide an output of +3/2 within the sandwich nodes and -1/2 in the region outside the nodes to produce a combined output of +1 within the sandwiches and -1 outside them. This same system can be viewed in several different ways, either as a system of three closed sandwiches that are flush together with two extra open sandwiches, or a single closed sandwich surrounded by two open sandwiches. Whichever interpretation is adopted suitable coherent output values of the various nodes exist. Values of +1 and 0 inside and outside of the positive sandwiches, while the negative sandwich in the middle with the values -1 and 0 inside and outside with outputs of 0 and -1 for the relevant open sandwiches, defines a suitable segmentation of the input space.

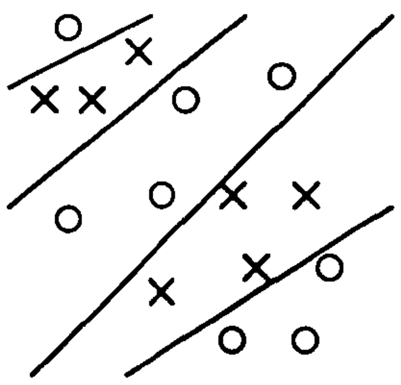


Fig 6.5 Segmentation of data using closed sandwiches

Subsymbolic reasoning in general sandwich systems

Given a sandwich representation of a training set, either specified or learned through a learning algorithm such as the ghost node training of chapter five, execution of the network can proceed as a well defined system of subsymbolic reasoning. Given the network in fig 6.6a we can ask a query about point A (fig 6.6b). The network execution reveals that an answer of +1 is given. The explanation of this solution is given via the presentation of the relevant coherent set of nodes, fig 6.6c and any training patterns that are part of this atomic knowledge element. More complicated cases where interacting elements exist are dealt with via a similar process, where the relevant interacting elements constructed from atomic elements are presented in the explanation.

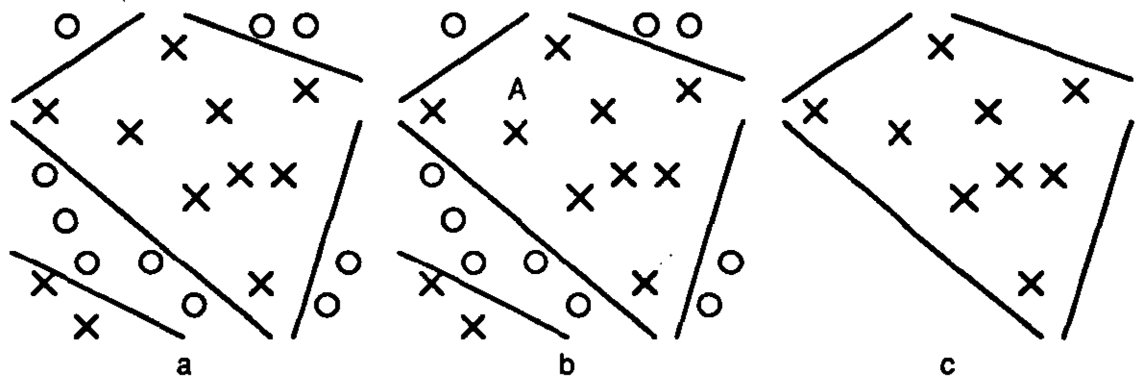


Fig 6.6 a. Segmentation of a set of data, b. Query of input A, c. Explanation region of query A

The structural constraints associated with sandwich nodes ensure that no complex Boolean interaction occurs between the sandwich nodes of each layer. This ensures that a linear combination of the sandwich nodes exist to provide the required output values. Therefore with the relevant sandwich node constraints only one layer of hidden nodes is required to model any problem. If explicit encapsulation of sandwich nodes is required as discussed in chapter five then an extra layer of hidden nodes are required. This is illustrated in fig 6.7 a & b, for the two input and multiple input case.

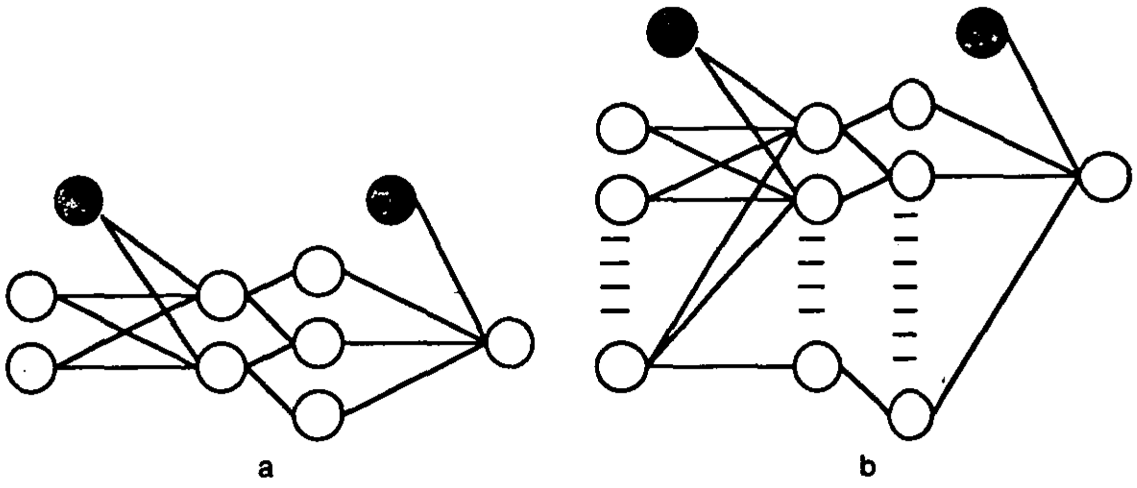


Fig 6.7 Explicit encapsulated sandwich nodes a. Two input case, b. Multiple input case

Explicit node encapsulation

Node encapsulation relies on the existence of a neural model of a region of the input space in which the output values are defined by the training set, while the output of zero is given for the remaining regions of the input space. A few simple examples of node encapsulation are examined.

The simplest case is when a single value of +1 or -1 is required on a single convex region of the input space. With a single node the activation function can be implemented via a node with a single input and a bias of 1.0. This has an input/ output relationship illustrated in fig 6.8. Similar activation functions for output values of -1 and different regional boundaries can also be implemented.

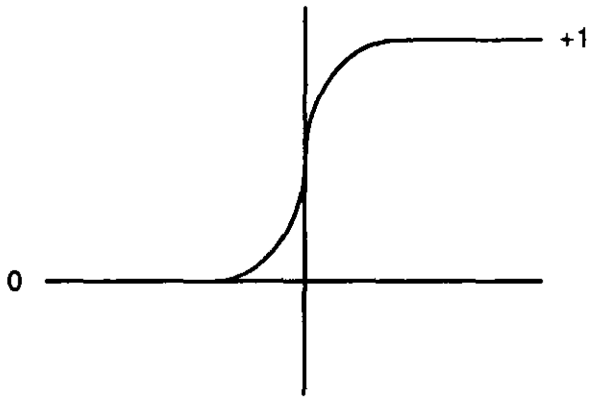


Fig 6.8 Input/ output relationship of node d, fig 6.9

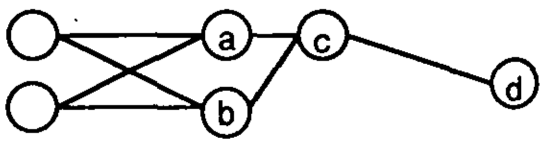


Fig 6.9 Explicit encapsulation of a sandwich node

An encapsulated sandwich node of a region defined by two lines can also be illustrated. Fig 6.10a shows the region of interest, while fig 6.9 shows a suitable network structure in which the node weights are defined as, node_a(0,1,1), node_b(0,1,-1), node_c(-1,1,1) and node_d(1,1).

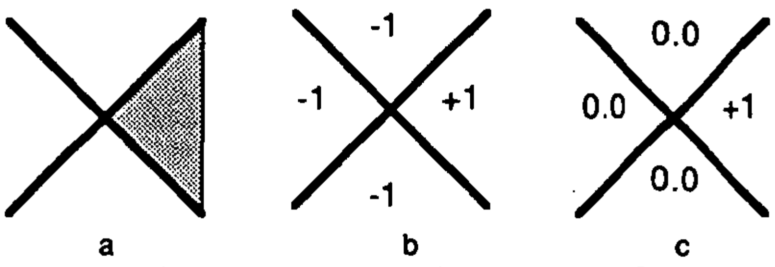


Fig 6.10 Sandwich segmentation of a region of the input space, a. Shaded region of interest, b. Output from a standard node, c. Output from the encapsulated sandwich node, making use of the activation function

The output of node_c is illustrated by the figure 6.10b which is defined over the whole input space and does not encapsulate the node function to the desired region. The output of node_d is shown in fig 6.10c, which is encapsulated to the desired region.

The function of a transformation over the whole input space can be constructed by using several disjoint encapsulated nodes that provide the correct output over the relevant region in which they contribute. An example in which a closed convex region is employed to construct a suitable encapsulated model of a problem is illustrated below.

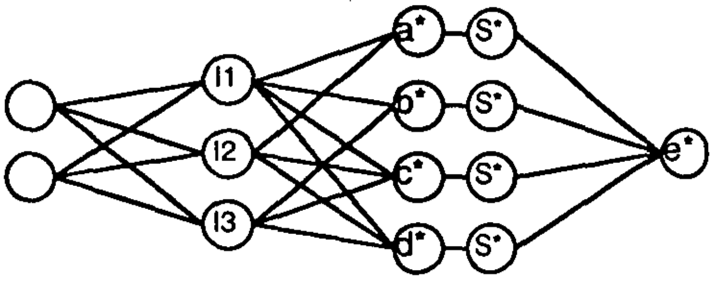


Fig 6.11 Neural network structure that employs triangular encapsulation and biplanar segmentation

Figure 6.11 shows the network structure used to model the problem while fig 6.12a shows the regions of the input space that are of interest. The weights in the neural model are defined by the input segmentation nodes node_I1(-1,-2,1), node_I2(-1,1,-2),node_I3(-1,1,1), the hidden decision region nodes node_a*(1,-1,-1), node_b*(1,-1,-1), node_c*(2,1,1,1), node_d*(1,-1,-1), the encapsulated sandwich nodes node_S*(1,1) and the final output node node_e*(0,1,1,1,1).

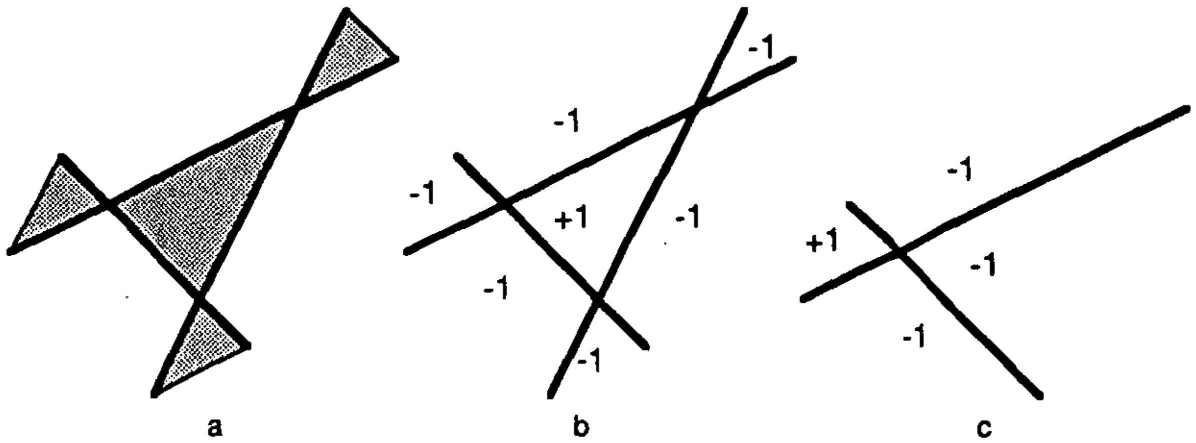


Fig 6.12 Segmentation of the input space into regions of interest, a. Shaded regions of interest, b. Output from an output node defined by three hidden nodes (lines in the input space) forming a triangular region, c. Output from an output node defined by two nodes (lines in the input space)

Figure 6.12 b & c show the standard output values of the decision nodes node_c and node_d respectively. These contribute in regions that we do not require and so an activation value must be applied. The output values from node_cS* is shown in fig 6.12d, while the output of the total system node_e* is shown in fig 6.12e. The output values for the other regions of the input space where the output is required to be -1 can be constructed in a similar manner. However if all of the remaining regions must provide an output of -1, we can achieve the outputs shown in fig 6.12f with a small negative bias applied to the output node, that is give the weights node_e(-0.01,1,1,1,1).

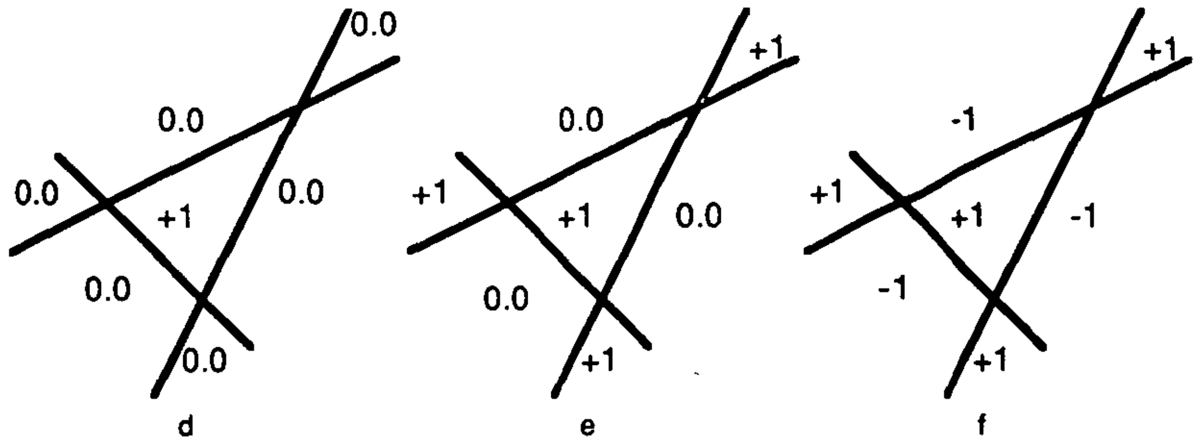


Fig 6.12 d. Encapsulated sandwich activity of triangular node, e. Activity of encapsulated system of a triangular node and three biplanar segments, f. A general system of activity

Putting knowledge into neural network models

Given a training set that is already modelled by a neural network or an extension of a set of data that has already been modelled, then automated backpropagation training techniques need not be employed. The knowledge available can be utilised to directly model the new training set. If known neural models do not exist for the subsets that form a disjoint partition of the new training set, then only a partial model can be constructed. The remaining components must be constructed via an automated training scheme. These problems are outlined below and illustrated by examples below.

There are three main mechanisms for deriving larger training sets from smaller ones. The first is that of the addition of input nodes so increasing the number of possible input patterns. The second is the expansion of the training set to cover a region of the input space that was not modelled before, that is the data consist of two or more subsets that can be modelled independently and amalgamated. The third and final mechanism for training set expansion is the general increased number of data points in the regions of the input space that are already modelled so leading to greater acuity in the neural model required. The first two cases can be dealt with via the amalgamation of suitable neural models of the subspaces and are discussed below. The third case requires automated training to be applied to extensions of the models that already exist so leading to models with greater acuity. This is discussed further.

Increasing input dimension

An n input data set can be viewed as two n-1 input data sets. That is the two sets define the values over the n-1 subspaces of the n dimensional input space. If neural models of the data sets of the n-1 subspaces exist then we can construct an n input neural model of the whole data set. This can be achieved via the use of encapsulated sandwich nodes that provide the required input values over the relevant subspace and an output value of zero over the remaining region. Fig 6.13 shows a neural model of one of the n-1 subspaces (in this case n = 5), a similar neural model will exist for the other n-1 subspace. The four input model's output node is defined by the weights $node\_O_4(b,w_1,w_2,w_3,w_4)$.

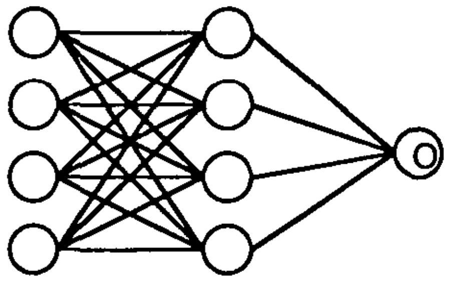


Fig 6.13 Neural model of a four input data set

For this example, the five input neural model must provide the correct output depending on whether the input data point is in the first four dimensional subspace or the second. Amalgamating the two submodels directly via summing the outputs and thresholding them is two simplistic and will only provide the correct output when the two submodels agree on the output over the four input nodes (fig 6.14a). If the two submodels provide a conflicting result the total output will be zero and no information can be gained.

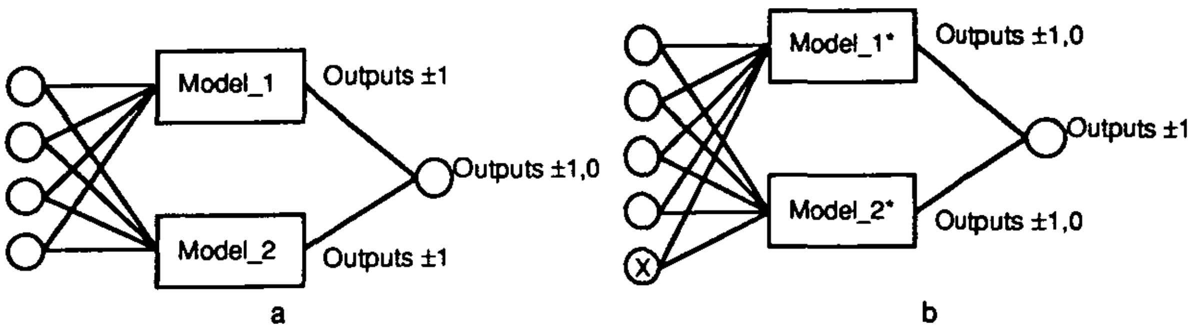


Fig 6.14 a. Simple amalgamation of models leading to invalid outputs, b. Encapsulated sandwich models amalgamated to produce valid output

Making use of encapsulated sandwich nodes that ensure the respective submodels give an output of zero over the region that they do not model ensures that a correct five input model is derived. This can in practise be achieved in two ways. The use of encapsulating and activating nodes in the final layer or in the hidden layer.

Inputs	Outputs
-1	-1
+1	0.0

Node A Inputs \ Other Inputs	-1	+1
-1	0.0	0.0
+1	-1	+1

Table 6.1 a. Output values of node A, the activation node used for the encapsulated sandwich nodes with one activation input, b. Output values of node B, the activated encapsulated sandwich nodes

The activation node used for this example is given by the node weights node_A(-1,1), whose output values are shown in table 6.1a. No output is given when the fifth input node (node X in fig 6.15a) has value +1, since the output of the submodel is required for this value. An output of -1 is given for activation when the input is -1 as this value is required to cancel with the activity of the whole system, which is -1 when the fifth input node has value -1.

The encapsulated sandwich node is defined by the weights node_B(0,1,-1), whose output values are shown in table 6.1b. The input values are the inputs to node_A and the output from the five input node system, node_O. (The weight values of the five input system are given by the weight matrix node_$O_5(b - w_5, w_1, w_2, w_3, w_4, w_5)$, where $2.w_5 > b + \Sigma^4_{i=1} |w_i|$). As can be seen from the output values of node_B, the encapsulated system has an output of zero for the region not modelled by the submodel and the specific output ±1 for the region modelled by the submodel.

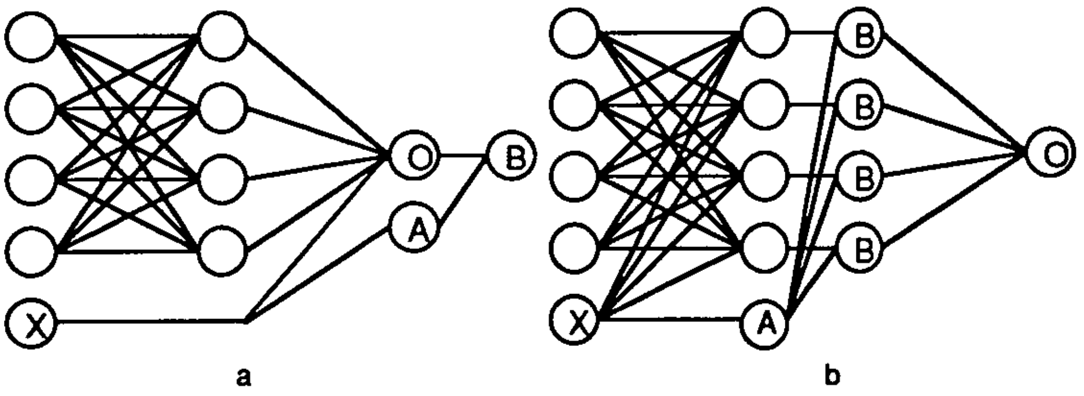


Fig 6.15 Neural model of half of the five input training set based on the four input model using a. Encapsulated sandwich nodes in the final layer, b. Encapsulated sandwich nodes in the hidden layer

The activated encapsulation nodes can be implemented in the hidden layer. Effectively each hidden node provides an output of zero when node_X has value -1, while providing the same output as the four node model when the input node node_X has value +1 (see fig 6.15b). Another amalgamation of subspace representations is considered in chapter seven that does not make use of the explicit encapsulation of the subspace activities. Instead the implicit nature of the subspace representation's interaction is exploited, implementing an amalgamation scheme that makes use of standard nodes in a single hidden layer.

Amalgamating regional models

The example above essentially solved the problem of amalgamating submodels of a problem where the submodels where defined over a reduced set of inputs, that is a lower dimensional space. The case where the submodels are defined over different regions of the same dimensional space can also be solved with a suitable amalgamation scheme. As before we must be able to produce activated encapsulated models of the subregions over which the given submodels are valid. This means that a trigger node or nodes are required which provide the output +1 if the submodel is valid over the given input point and -1 if the submodel is not necessarily valid.

Second Input / First Input	-1	+1
-1	-1	-1
+1	-1	0.0

Table 6.2 Output values of node C, the activation node used for the encapsulated sandwich nodes with two activation inputs

We can illustrate this with the example below. Take the four input network model of fig 6.13 to be the given submodel of a region of the input space. Two nodes node_Y and node_Z are the activating inputs whereby if both node_Y and node_Z give the output +1, the output of the given submodel is valid. The structure of the activation illustrated in table 6.2 is given by the weights node_C(-1, 1,1). This is the generalisation of activation node_A to two inputs.

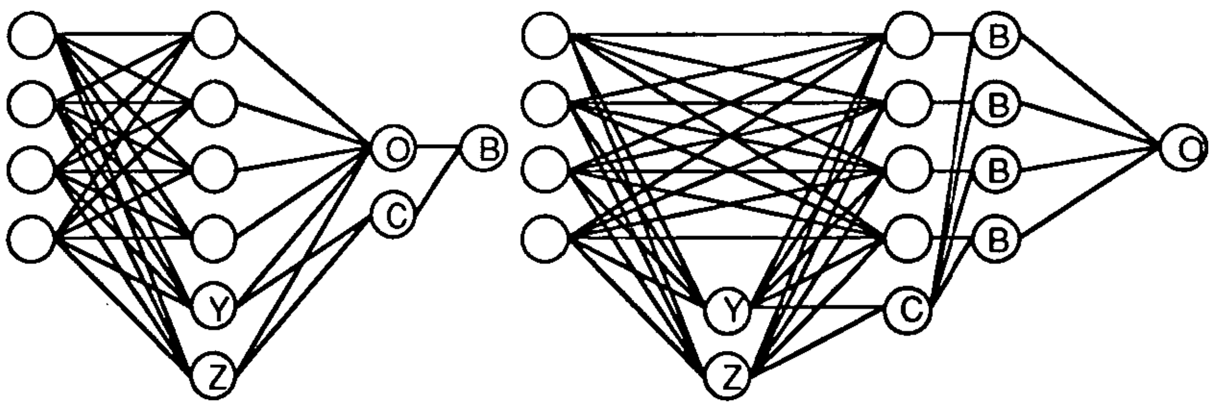


Fig 6.16 Neural model of a subregion of the training set defined by the nodes Y and Z, based on the model of the subregion, using a. encapsulated sandwich nodes in the final layer, b. encapsulated sandwich nodes in the hidden layer

Fig 6.16a shows the case where the activated encapsulated sandwich node is applied to the output node. The output node node_O is analogous to the example above, (The weight values of the system are given by the weight matrix node_O(b-w_5- w_6,w_1,w_2,w_3,w_4,w_5, w_6), where $2.(w_5 - w_6) > b + \Sigma^4_{i=1} |w_i|$). This system provides the required output over the specified region and zero everywhere else, so allowing it to be immediately amalgamated with models of disjoint subregions of the input space. The structure of the model using encapsulated sandwich nodes in the hidden layer is shown in fig 6.16b. The construction of these models are again analogous to the example above with the node_C acting as the activation node for the sandwich nodes.

Neural model development

The final case to be considered is when a training set is enhanced by adding more data points in the region that has already been modelled by the neural network. In this case the data points that have been added are either correctly or incorrectly modelled by the old model. If the new points are correctly modelled by the old network then no improvement can be made or need be made. New points that are incorrectly modelled contribute to improving the neural model. Since no separate model of the new data points exist a solution employing an amalgamation technique is not possible. An automated training system can be employed making use of the existing model. As the experiments of appendices A4 and D show this may not converge to optimal solutions and often the previous trained model is lost. To overcome the problem of convergence a larger network can be employed, with its greater representational power.

Reliability of encapsulated sandwich schemes

The reliability of a sandwich neural network model can be examined in a similar manner to the analysis of standard neural techniques based on the training sets employed. A trained sandwich neural model contains the training set and the regions of the input space that these data points occupy. There are essentially two situations to consider. Firstly the unknown data point falls in the region that is modelled, if this occurs, we can identify the region, the encapsulated nodes that fire and the members of the training set that are characteristic cases for the given input region, whether the unknown data point is a training point or not. That is;

Probability(correct output|member of modelled region) = 1.

If the unknown data point is not part of the modelled region of the input space no information is available about the value that the point should take and so;

Probability(correct output|not member of modelled region) = 1/2.

Even this final uncertainty in network behaviour can be removed by ensuring that all regions not modelled by the network explicitly have a specified output value of either +1 or -1. That is the whole of the input space is essentially modelled by the network and so becomes perfectly predictable.

Summary

This chapter introduced the sandwich node as the atomic element of a monotonic neural network knowledge representational scheme. It was shown that the function of a sandwich node or system of nodes could be encapsulated in a single node that only contributed to the decision in a well defined region of the input space. This was illustrated with several simplified examples. Finally the reliability and predictability of encapsulated sandwich systems were seen to be greater than standard neural network techniques, since the interaction between the hidden nodes of the sandwich systems were well structured and so minimised.

Part III. Designing Neural Network Systems

Chapter 7. Network Size and Topology

Introduction

The number of input nodes and output nodes and the transformation being modelled influences the number of hidden layers and size of these layers needed in the feedforward neural network.

The number of input and output nodes is a function of the application domain itself and so will be known. The transformation being modelled is typically unknown unless specialised knowledge of the domain exists. Constructing a suitable network becomes a question of knowing how many layers and of what size are required to model a possibly unknown transformation, given the number of input and output nodes. The hidden structure of the network between the input and output nodes that is needed to model a given transformation is investigated in this chapter.

Existing results

Lipmann('87), studying a four layer network (that is one with two layers of hidden nodes) suggests that at least three times the number of nodes in the second hidden layer is required for the first hidden layer. Lipmann argues that the second hidden layer would have as many nodes as the number of disjoint decision regions in the input space where a decision region is an area in the input space with a specified output. Each disjoint region of the input space would require a node in the second hidden layer to recognise whether the input case was in that region. The output can then be calculated for that particular input case. Each disjoint decision region will be generated by at least three lines and so there will be three times the number of nodes in the first layer as the second (see region A of figure 7.1). Lipmann's limit is an upper bound as shown by the simple addition of one more line to figure7.1 creating at least one more disjoint area, region B.

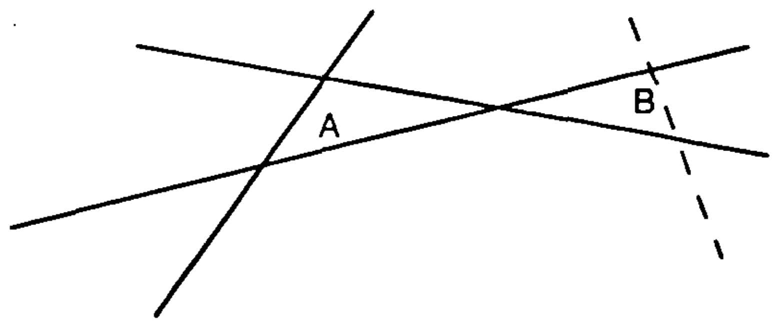


Fig 7.1 Three lines are required to enclose an area A, however an extra area can be created by adding one more line giving area B

Mirchandani et al('89) present some more results on the node requirement of the hidden layer based on the number of linearly separable decision regions that exist in the space in question. This limit provides a measure of the size of the feedforward network required to model a particular problem, given the number of training examples. The result presented is that $M(H,n)= \Sigma^{n}_{k=0} {}^{H}C_{k}$ and ${}^{H}C_{k}= 0$, if $H< k$, where $M(H,n)$ is the maximum number of decision regions possible with H hidden nodes in an n dimensional space and ${}^{H}C_{k}$ the standard binomial coefficient. This result gives an idea of how many independent training points can be learned. To apply this analysis to the general case however, where the decision regions are not necessarily known, is impossible. Therefore a general limit on the size requirement of the hidden layer can not be provided from this result.

Huang et al('91) provide another comprehensive study on the number of hidden neurons required in feedforward systems. They approach the problem in the same manner as Mirchandani et al('89). That is, they consider the size of the training set and the number of nodes required to model that set. For the special case of n dimensional Boolean functions with one output they present the result that an $\{n^{*} \lfloor((m+ 1)/ 2)\rfloor+ 1\}$ element training set can be modelled by a feedforward net with m hidden nodes. ($\lfloor x \rfloor$ is the largest integer less than or equal to x). For a three input net we have a total Boolean training set of eight elements giving a value for m of five. It can be easily verified by exhaustive search that only three hidden nodes are required to model any three dimensional Boolean function. The work of Mirchandani et al and Huang et al are examined in the context of real valued inputs in chapter eight.

Hertz et al('91) discuss size limitations of feedforward systems, and offer the single hidden layer limit, as does Hinde('90), for the case of Boolean transformations. The question of the size of this hidden layer is not discussed in depth by Hertz and the weak limit of 2^n, where n, the number of input units, is given. This is the network in which each input case has its own individual hidden unit to recognise it.

Properties of feedforward Representations

In studying neural networks several interesting representational properties have been encountered. The first is that for some particularly simple transformations, very few hidden nodes are needed. For instance, if the output is dependent on only two of the input nodes, then even if we have a k dimensional problem, where k is a large number, only two hidden nodes are required. A seemingly complex transformation like that of parity requires only k hidden nodes for the k input case. (The modelling of the parity problem is discussed in chapter five). The second point of interest is that there are many network representations of each transformation, many of which will require a large number of hidden nodes. For instance the parity case can be modelled in as many as 2^k different ways using just k hidden nodes, while it can also be modelled by a net which has hidden nodes that isolate a single input example each. This case will have 2^{k-1} hidden nodes for the k input case.

Splitting and projecting nodes

Given that the nodes represent hyperplanes in the k dimensional input space, they can be projected onto any other hyperplane in the space. The hyperplanes formed by considering any input node and fixing the nodes value at +1 or -1 defines two distinct hyperplanes in the k dimensional input space. Fig 7.2 a, b, c, d, e shows how fixing an input node reduces the dimensionallity of the given node.

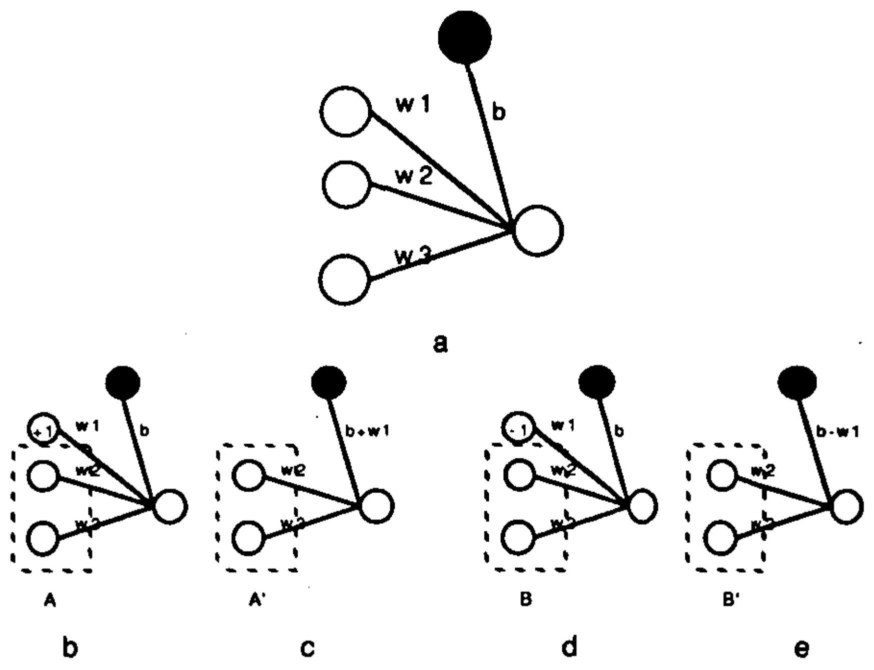


Fig 7.2 a. A three input node. Splitting a node into its component parts by fixing one of the inputs at, b. +1 or, d. -1, c. & e. Lower dimensional network that can emulate the original network

The projections of the hidden node hyperplanes onto the input node hyperplanes can be classified into three groups. See fig 7.3 for the original hyperplanes and fig 7.4 for their respective projections.

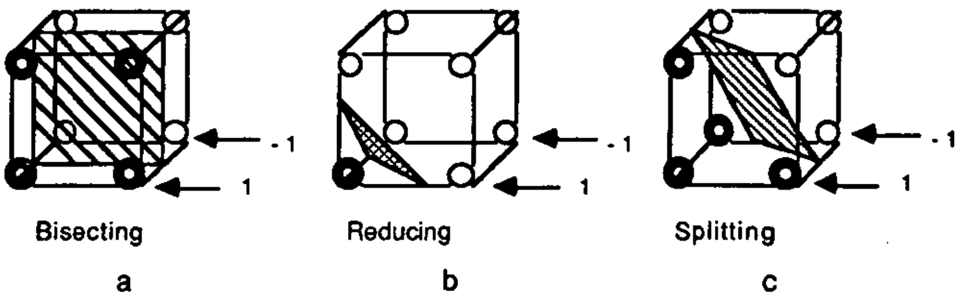


Fig 7.3 Three general forms of nodes viewed with respect to a given input axis. a. Bisecting, b. Reducing, c. Splitting. The projecting planes are denoted by 1 and -1

These three groups are respectively bisecting hyperplanes, reducing hyperplanes and splitting hyperplanes. Bisecting hyperplanes can also be represented by the hyperplane input node= 0, and so do not make a contribution to the points in the subspace other than as an external bias. (Fig 7.4a). Reducing hyperplanes have a contributing projection in only one subspace and in the other only offer an external bias. (Fig 7.4b). Splitting planes have a contributing projection into both subspaces, input node= +1 or -1. (Fig 7.4c).

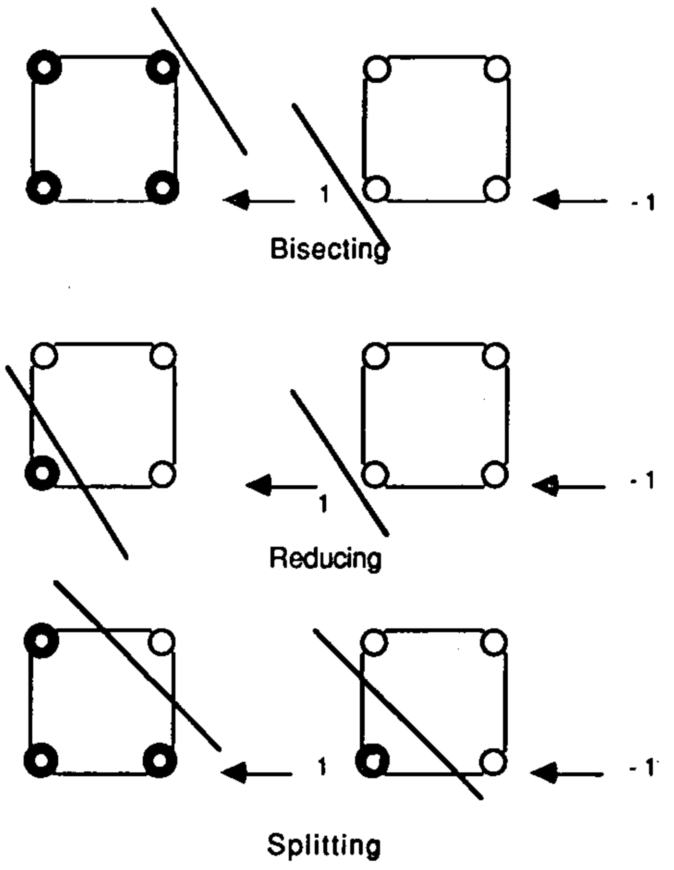


Fig 7.4 Views of the three types of nodes after splitting along an axis, a.Bisecting, b. Reducing, c. Splitting

The analysis can proceed in a similar manner for nodes with a general number of inputs. (Fig 7.5a).

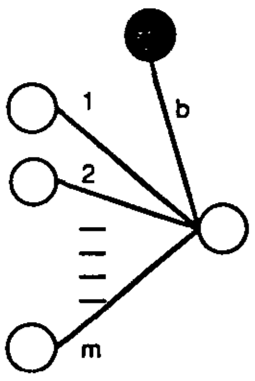
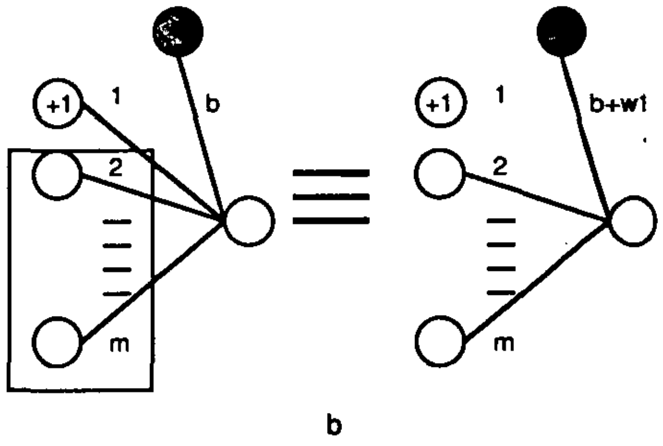


Fig 7.5 a. A general m input node

By fixing an input node at either +1 or -1 the m dimensional node can be split into two m-1 dimensional cases. The two splits of the m dimensional nodes are equivalent to the two m-1 dimensional nodes as seen in fig 7.5 b & c.



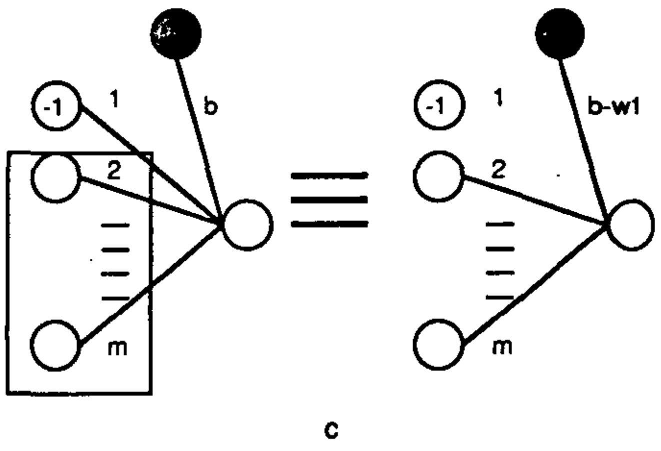


Fig 7.5 The equivalence of two splits to the component nodes, the fixed input node acting as a contribution to the bias of the projected node, b. Setting the node value at +1, c. Setting the node values at -1

Constructing network representations

Methods for constructing neural networks were briefly discussed in chapter 1, but they do not make use of the large body of knowledge that may exist about the transformation being modelled. Huang et al('91)'s technique, makes specific use of the knowledge of the training data to construct the network required to model the transformation. It does not make use of any larger knowledge structures that may already exist, such as representations of subspaces of the transformation (see chapter six). Techniques that amalgamate knowledge of lower dimensional subspace representations into representations of the total transformation are presented below.

Amalgamation of representations

Neural representations of a transformation can be constructed from the lower dimensional representations that may already exist. This is achieved by amalgamating the lower dimensional schemes into an overall higher dimensional representation. At the simplest level, assume an n input transformation exists whose splits about a single input node, that is two n-1 input transformations have neural representations, as shown in fig 7.6 a & b.

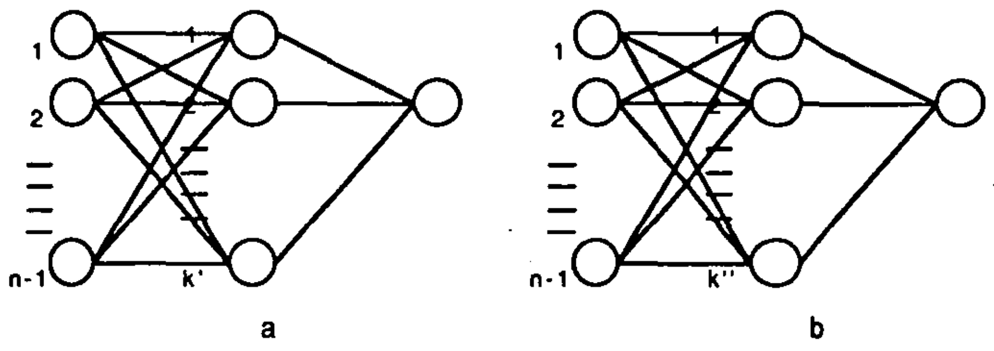


Fig 7.6 Representation of the two n-1 input subsets of the full transformation, a. The split input node= +1, b. The split input node= -1

The two subset representations are defined by the weights on the nodes in the hidden layers and the output layer, that is, w_{ij}', $1\leq i\leq k'$, $0\leq j\leq n$ for the hidden layer and w_i', $0\leq i\leq k'$, for the output node of the first split and also w_{ij}'', $1\leq i\leq k''$, $0\leq j\leq n$ for the hidden layer and w_i'', $0\leq i\leq k''$, for the output node of the second split. The weights with subscripts of zero refer to the bias weights. We can construct a higher dimensional representation making use of two barrages of nodes from the lower dimensional representation. This is shown in fig 7.7.

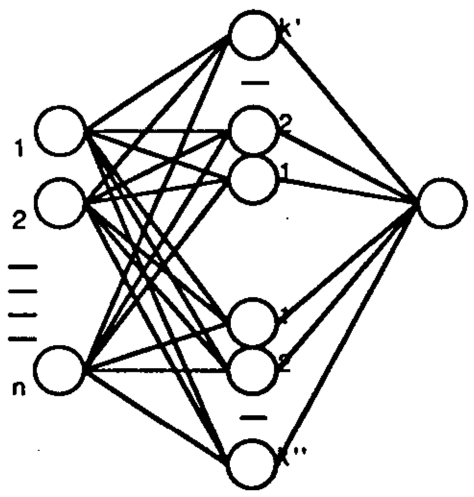


Fig 7.7 Representation of the total n input transformation

This amalgamated representation is defined by the weights, w_{ij}, $1\leq i\leq k'+k''$, $0\leq j\leq n$ for the hidden layer and w_i, $0\leq i\leq k'+k''$, for the output node. The weights in this representation all correspond to weights in the two subspace representations with the exception of the bias weights and the weight on the n^{th} node, about which the representation was split.

The weights that are unaffected are as follows;

$w_{ij}= w_{ij}'$, $1\leq i\leq k'$, $1\leq j\leq n-1$ and

$w_{ij}= w_{ij}''$, $k'+1\leq i\leq k'+k''$, $1\leq j\leq n-1$ for the hidden layer and

$w_i = w_i'$, $1\leq i\leq k'$ and

$w_i = w_i''$, $k'+1\leq i\leq k'+k''$, for the output node.

The as yet undefined weights can be calculated by examining the behaviour of the n input representation with the behaviour of the subset representations. This is done by setting the n^{th} input node, about which the problem is split, at +1 and -1 .

The condition that each barrage of nodes only contributes in a discriminatory manner when the n^{th} input node is set at either +1 or -1 but not both is required. Adding the condition that the nodes output -1 over the region that they do not contribute, gives us the following equations over the weights of nodes in the hidden layer. It should be noted, the substitution $b_i' = w_{i0}'$, $b_i'' = w_{i0}''$, $b_i = w_{i0}$, for $1 \le i \le k' + k''$, has been made in the following equations to make clear which weights are the bias weights.

threshold($b_i' - w_{in}' + \Sigma_{j=1}^{n-1} w_{ji}' * X_j$)= -1, for all input vectors X.

threshold($b_i'' + w_{in}'' + \Sigma_{j=1}^{n-1} w_{ji}'' * X_j$)= -1, for all input vectors X.

These equations give us

$b_i' - w_{in} < -\Sigma_{j=1}^{n-1} |w_{ji}'|$ for $1 \le i \le k'$ and

$b_i'' + w_{in} < -\Sigma_{j=1}^{n-1} |w_{ji}''|$ for $k' \le i \le k' + k''$.

Since the only unknown is w_{in} it can be specified to satisfy the equations above for $1 \le i \le k' + k''$.

The fact that over the regions that the barrages of weights contributes in a discriminatory manner, the output of the n dimensional net is identical to that of the sub representations, gives us the following equations.

$\{ b_i + w_{in} + \Sigma_{j=1}^{n-1} w_{ji} * X_j\} = \{ b_i' + \Sigma_{j=1}^{n-1} w_{ji}' * X_j\}$ for $1 \le i \le k'$ and

$\{ b_i - w_{in} + \Sigma_{j=1}^{n-1} w_{ji} * X_j\} = \{ b_i'' + \Sigma_{j=1}^{n-1} w_{ji}'' * X_j\}$ for $k' \le i \le k' + k''$.

These equations simplify to

$\{ b_i + w_{in}\} = b_i'$ for $1 \le i \le k'$ and

$\{ b_i - w_{in} \} = b_i''$ for $k' \le i \le k' + k''$.

Since the only unknown is b_i, its value can be calculated for $1 \le i \le k' + k''$.

Having specified all the bias values and weight values from the n^{th} input node of all the nodes of the hidden layer, the bias weight of the output node can be calculated. Again, the substitution $b_{out} = w_0$, $b_{out}' = w_0'$, and $b_{out}'' = w_0''$, is made in order that the bias weight can be clearly distinguished. The nth input node of the n dimensional representation is set at +1 and -1 and its behaviour compared to that of the sub representations. The following equations are obtained.

$\{ b_{out} + \Sigma_{j=1}^{k'} w_{ji} * X_j - \Sigma_{j=k'+1}^{k'+k''} w_{ji} \} = \{ b_{out}' + \Sigma_{j=1}^{k'} w_{ji}' * X_j \}$ for the case nth input node= +1 and

$\{ b_{out} - \Sigma_{j=1}^{k'} w_{ji} + \Sigma_{j=k'+1}^{k'+k''} w_{ji} * X_j \} = \{ b_{out}'' + \Sigma_{j=1}^{k''} w_{ji}'' * X_j \}$ for the case nth input node= -1.

These equations reduce to

$\{ b_{out} - \Sigma_{j=k'+1}^{k'+k''} w_{ji} \} = b_{out}'$ for the case nth input node= +1 and

$\{ b_{out} - \Sigma_{j=1}^{k'} w_{ji} \} = b_{out}''$ for the case nth input node= -1.

The only unknown is b_{out} and the equations are consistent if

$\{ b_{out}'' - \Sigma_{j=k'+1}^{k'+k''} w_{ji} \} = \{ b_{out}' - \Sigma_{j=1}^{k'} w_{ji} \}$, which can be achieved by a suitable scaling one set of weights. Therefore we can provide a consistent value for b_{out}.

Amalgamation of nodes

The method for constructing networks by amalgamating low dimensional representations will produce representations that are far from minimal. The method treats each subspace independently, not exploiting the similarities that may exist within the representations chosen. Several methods are presented that will reduce the size of the network representations. This will be achieved by amalgamating nodes that do not provide any extra information in a representation.

Two nodes can be amalgamated in a particular layer that contribute identical outputs over the set of inputs and create a new node which produces this required output (the conditions under which this is possible is discussed below). Nodes in the next layer that were connected to either of the original nodes are connected to the new node using their original weights. If the node in the next layer was connected to both of the original nodes then the new weight is the scalar sum of the original weights.

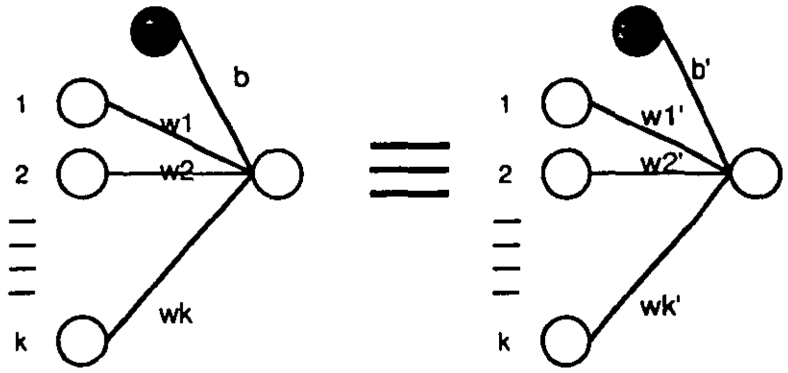


Fig 7.8 Nodes are equivalent if the weights vectors are scalar multiples,(b,w1,w2,..,wk)= c*(b', w1',w2',..,wk'), where c is a scalar constant

This amalgamation scheme is useful since it offers us the possibility of reducing the size of representations that are constructed. The difficulty in making use of this property is that of recognising if two nodes are synchronised over the set of inputs. For large input spaces and a large number of nodes it would be impractical to verify if two nodes are synchronised.

Examining the weight space that defines the nodes allows us to decide whether two nodes are synchronised without observing the behaviour of the nodes over all the input cases. Let us consider two nodes, node' and node". If node' and node" are defined over the same inputs and have identical weights or identical weights to a scalar factor, then the two nodes are synchronised (see fig 7.8). This follows directly from the node formula.

If the weights are perturbed slightly the two nodes may still be synchronised. That is, if node N and node N' have similar weights, they may be synchronised. We examine the conditions under which two nodes are synchronised.

To ensure that the weight vectors are scaled equally the largest weight of node N is selected, w_1 say, and the weights of node N' scaled so that $w_1 = w_1'$. The perturbation of the i^{th}

weight, Δw_i, is defined as the difference between the i^{th} weight of node N and node N'. That is $\Delta w_i = w_i' - w_i$. The weight vectors being similar, a new node, node N", can be constructed by selecting one of the weights w_i or w_i' for each component w_i''. This new node will be synchronised with node N and node N' if the sum of the possible perturbations, $\Sigma_{i=1}^{n}|\Delta w_i|$, is small enough to preserve the output. Since the output is **threshold**($\Sigma_{i=1}^{n} w_i * X_i$), where **threshold**(X) = +1, for all X > 0, **threshold**(X) = -1, for all X < 0, we have the condition that;

if $|\Sigma_{i=1}^{n} w_i * X_i| > \Sigma_{i=1}^{n}|\Delta w_i|$, and $|\Sigma_{i=1}^{n} w_i' * X_i| > \Sigma_{i=1}^{n}|\Delta w_i|$, for all input values X, then the nodes node N and node N' are synchronised. The node N and node N' can be amalgamated into node N", so reducing the size of the neural representation.

This requires us to check the output behaviour of the nodes over the whole input set. Again, this may not be practical for large training sets. By ensuring that $|\Sigma_{i=1}^{n} w_i * X_i| > c$ and $|\Sigma_{i=1}^{n} w_i' * X_i| > c$, for all input values, where c is a non zero positive constant, during the training of the network, the problem is reduced to that of checking that $c > \Sigma_{i=1}^{n}|\Delta w_i|$. If this is the case the nodes are synchronised and so can be amalgamated.

Amalgamation of nodes in different barrages

When two nodes contribute in a discriminatory manner over two distinct regions of the input space it may be possible to amalgamate them into a single node. This new node will be synchronised with the original nodes over the regions that they are not null. The case where we have nodes from different barrages represents the simplest possible case. Nodes in one barrage only contribute in a discriminatory manner over half the input set. One barrage fully represents the problem when a particular node is set at +1 while the other does so when the node is set at -1. The barrages together, fully represent the total problem.

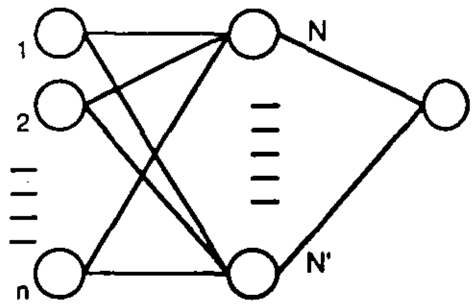


Fig 7.9 Two nodes in different barrages

As was the case with the amalgamation of the synchronised nodes in the previous section, two nodes from different barrages that have weight vectors that are identical over the independent input nodes examined (see fig 7.9 and fig 7.10 a & b). These nodes are defined by the biases b and b' and weights w_i and w_i', where $w_i = w_i'$, for $1 \leq i \leq n-1$. The bias nodes and nth input node weight are in general distinct since these would have been specified by the technique of amalgamating the representations as discussed above.

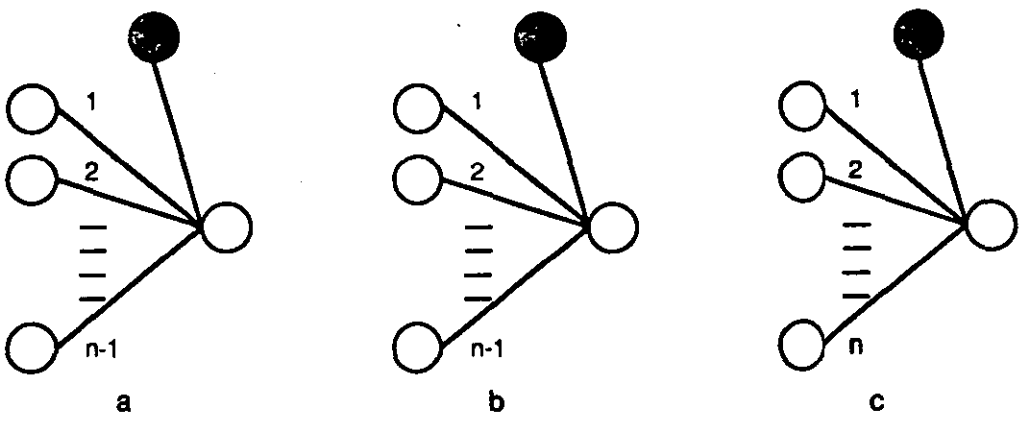


Fig 7.10 The two n-1 input nodes that contribute in different splits about the nth input node, with their amalgamated node defined over n input lines, a. node N, b. node N', c. amalgamated node

If a node N" that represented the whole problem existed (fig 7.10 c), defined by the bias b and weights w_i then it would have to satisfy the following equations.

$$\{ b'' + w_n'' + \Sigma_{j=1}^{n-1} w_j'' * X_j\} = \{ b + w_n + \Sigma_{j=1}^{n-1} w_j * X_j\},$$

$$\{ b'' - w_n'' + \Sigma_{j=1}^{n-1} w_j'' * X_j\} = \{ b' - w_n' + \Sigma_{j=1}^{n-1} w_j' * X_j\}$$

we have that $w_i'' = w_i = w_i'$, so

$$b'' + w_n'' = b + w_n,$$

$$b'' - w_n'' = b' - w_n'.$$

Solving these simultaneous equations, unknowns are b and w_n, gives

$$b'' = (b + b' + w_n - w_n')/2,$$

$$w_n = (b - b' + w_n + w_n')/2.$$

Therefore the two nodes can be amalgamated into a single node, reducing the overall representational scheme. In general the condition that $w_i = w_i'$, for $1 \le i \le n-1$, is not required but that (discounting the effect of the bias and nth input weight) the nodes are synchronised.

Carrying through the analysis from the amalgamation of nodes above, if $|\Sigma_{i=1}^{n-1} w_i * X_i| > \Sigma_{i=1}^{n-1}|\Delta w_i''|$, and $|\Sigma_{i=1}^{n-1} w_i' * X_i| > \Sigma_{i=1}^{n-1}|\Delta w_i''|$, for all input values X, a single amalgamated node can be constructed.

And similarly by ensuring that $|\Sigma_{i=1}^{n-1} w_i * X_i| > c$ and $|\Sigma_{i=1}^{n-1} w_i' * X_i| > c$, for all input values, where c is a non zero positive constant, during the training of the network means that if $c > \Sigma_{i=1}^{n-1}|\Delta w_i|$, a single amalgamated node can be constructed.

Definitions

Some terms are defined below which will be used in the following sections.

Minimal representations

We can create many different networks to represent an individual transformation. The smallest network that can represent a transformation is called the minimal representation. The number of nodes in this network is the minimum number of nodes required to model the transformation in question.

Oversupplied networks

A network representation that has more nodes than the minimum number required to model the transformation in question, is capable of modelling the transformation. Given such a network, training can begin. Back propagation can be applied over the training data and after the convergence of the training algorithm, the network will represent the transformation. A network which is not minimal with respect to a given transformation and has more nodes than the minimal representation is said to be oversupplied.

Undersupplied networks

Conversely, if we try to train a network with fewer nodes than the minimal representation, then the training algorithm will not converge. The network is incapable of representing the particular transformation and no amount of training will overcome the problem. Such a network is said to be undersupplied.

Worst transformation

Given a specific number of input nodes n say, and a single output node, we have a possible $_2(2^n)$ transformations that we can choose from. Different transformations will have different minimal representations. One or more transformations will have a minimal representation that is at least as large, if not larger than all the other minimal representations. This transformation is called the worst transformation and will be referred to as the worst case.

Fullness

A representation is said to be full if it uses at least as many hidden nodes as would be required to model the worst problem for that particular number of input nodes. This means that a full representation is capable of representing any transformation, it is just the training algorithm that perturbs the hidden nodes to produce a particular representation of the problem in question. A full representation will be oversupplied if it is not modelling the worst transformation.

Tightness

A representation is said to be tight if it uses as many but no more hidden nodes than would be required to model the worst problem. A tight representation is not oversupplied when modelling the worst case for that particular number of input nodes. A tight representation will be oversupplied when modelling a transformation that is not the worst case.

Null nodes

If a node or set of nodes produces an output that is identically +1 or -1 over a specific set of input examples, then that node or nodes is said to be null or null contributing over that input set. (Huang et al '91). A null node does not contribute in a discriminatory manner over the input set and so can be considered to be an external bias rather than a contributing node.

Representational definitions

More terms which are relevant in discussing neural network models of specific transformations are defined below.

i. A transformation is said to be k-representable if a neural net with k-hidden nodes can represent the problem.
ii. Two nodes defined over the same inputs are said to be consistent the outputs are synchronised.
iii. A node defined over a subset of inputs is said to s-consistent with a node defined over the whole set of inputs if in the subspace of the input space, the second node is consistent with the first.
iv. Two nodes are said to be m-consistent if there exists a node, possibly defined over a

larger set of inputs, such that these two nodes are s-consistent with it.

v. Two planes in two k-1 dimensional projected spaces are 1-consistent if there exists a plane in the k dimensional space such that on projecting along the particular axis the splitting planes produced are the ones in question. Therefore if two k-1 dimensional nodes are 1-consistent, we can construct a k dimensional node which when projected along the required axis provides the two nodes in question. The k dimensional node which is constructed is s-consistent with the two k-1 dimensional nodes. The two 1-consistent k-1 dimensional nodes are themselves m-consistent.

vi. Two planes in two k-1 dimensional projected spaces are 2-consistent if there exists a plane in the k+1 dimensional space such that on projecting along the two axes the splitting planes so formed are the ones in question. Therefore if two k-1 dimensional nodes are 2-consistent, we can construct a k+1 dimensional node which when projected along the required axes provides the two nodes in question. The k+1 dimensional node which is constructed is s-consistent with the two k-1 dimensional nodes. The two 2-consistent k-1 dimensional nodes are themselves m-consistent.

Topological limits on feedforward nets

Number of layers required

In this section we will only consider Boolean transformations to a single binary output, i.e. f: $B^n \rightarrow B$ (see fig 7.11). The extension of the result to multiple output nodes is natural, since each output can be considered independently and so does not require any extra layers.

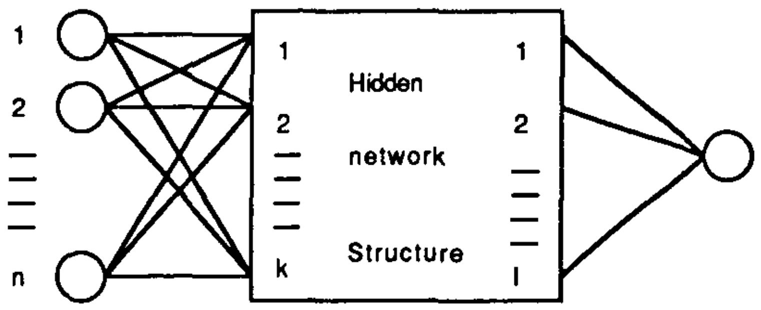


Fig 7.11 Structure of a general neural network with a single output

This transformation f: B^n->B , can be represented by a logical function of the input units. This transformation can be converted into conjunctive normal form or conjunctive canonical form as it is sometimes called, using just the standard logical NOT, AND and OR operators (Birkhoff et al '70 & '77).

Any Boolean transformation can be put into conjunctive normal form (Birkhoff et al '70). This is derived from the fact that all Boolean transformations are generated by the operators conjunction, disjunction, negation and the atomic propositions. Conjunctive normal form is the representation of a Boolean formula, using just the conjunction and disjunction operators, in which the formula is just a conjunction of disjunctions. The operators AND and OR satisfy the distributive equations;

a AND (b OR c)= (a AND b) OR (a AND c) and

a OR (b AND c)=(a OR b) AND (a OR c).

and the conversion operation under NOT;

NOT(a AND b)= (NOT(a) OR NOT(b)),

NOT(a OR b)= (NOT(a) AND NOT(b)).

It can be shown that any Boolean formula of conjunctions, disjunctions negations and atomic propositions can be transformed into conjunctive normal form using the formula transformations above.

An example of a Boolean formula over the propositions, a, b, c, d and e in conjunctive normal form is;

(a OR b OR NOT (c)) AND(b OR NOT (d)) AND(a OR e) AND(NOT (c) OR d OR e)

Therefore the conjunctive normal form can be represented by a network with two layers of weights, i.e. one with just one hidden layer of nodes (see fig 7.12). And so we derive the fact that any Boolean transformation f can be modelled by a feedforward net with at most one hidden layer of nodes.

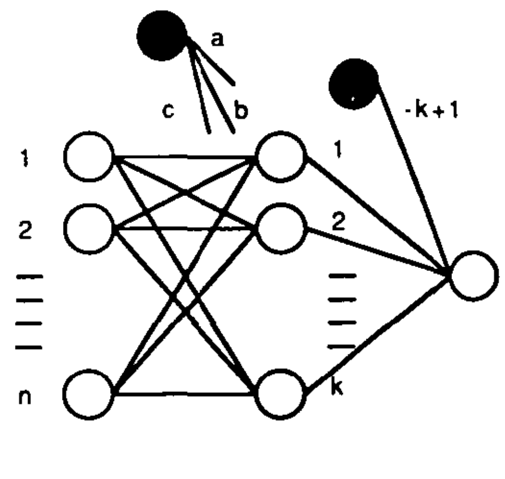


Fig 7.12 Neural net representation of conjunctive normal form of a transformation

The general Boolean transformation f: B^k->B^p can be considered as p distinct Boolean transformations f: B^k->B which can be modelled with a single layer. Therefore the multiple output case also requires only one hidden layer, with the output nodes sharing hidden nodes when the conjunctive normal form possesses identical terms (see fig 7.13). In fig 7.13, and other complex diagrams of network structure, the bias node is not shown. Suppressing the bias nodes allows the number of nodes and interconnection pattern to be viewed unobstructed.

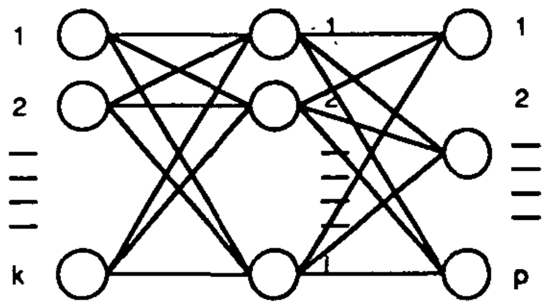


Fig 7.13 Neural net with a single hidden layer

Size of the hidden layer

In this section we will restrict ourselves to transformations with just one output unit, i.e. f: B^n->B. The extension of the result to the multiple output case will be presented in the next section.

We will prove the theorem that for a neural net of n input nodes and one output node a maximum of n nodes are required in the hidden layer to fully represent any transformation.

Some problems easily satisfy the theorem, while general transformations don't easily lend themselves to the same analysis. Enumerating each transformation and its network representation is impractical for large dimensional transformations.

Theorem All Boolean transformations with n inputs can be modelled by at most n hidden nodes.

Proof

	a.	output
A	+ 1	+ 1
	- 1	+ 1

	b.	output
A	+ 1	- 1
	- 1	- 1

	c.	output
A	+ 1	+ 1
	- 1	- 1

	d.	output
A	+ 1	- 1
	- 1	+ 1

Table 7.1 Enumeration of the possible Boolean transformations with a single input

The theorem is true for n= 1, 2, and 3 by enumerating the possible Boolean transformations and showing that they can be modelled by networks satisfying the hypothesis. table 7.1 shows the four possible transformations for n= 1. They can all be modelled by a single node, so having no hidden node trivially satisfy the hypothesis. For n= 2, the sixteen possible transformations are seen in table 7.2 . All these can be solved with a single node, without any hidden layers except the two exclusive OR cases (case d table 7.2).The worst case is when we have the exclusive OR type problem and this can be solved with two nodes (see fig 7.14a), i.e. two discriminating lines. It also clear that by enumerating all the sixty four possible cases, that the conjecture is true for n= 3. Most satisfy the theorem trivially, the exclusive OR or parity in three dimensions, providing the worst case, which can be modelled with three hidden nodes (see fig 7.14b).

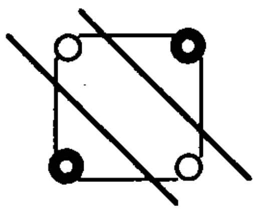
		B	
	a.	+ 1	- 1
A	+ 1	+ 1	+ 1
	- 1	+ 1	+ 1

		B	
	b.	+ 1	- 1
A	+ 1	+ 1	+ 1
	- 1	+ 1	- 1

		B	
	c.	+ 1	- 1
A	+ 1	+ 1	+ 1
	- 1	- 1	- 1

		B	
	d.	+ 1	- 1
A	+ 1	- 1	+ 1
	- 1	+ 1	- 1

Table 7.2 Enumeration of the possible Boolean transformations with two inputs. The sixteen possible transformations are given by negating the output or input lines of the four cases above, Case a. and its negated outputs gives two distinct transformations, Case b. and its negated inputs and outputs gives a total of eight distinct transformations, Case c. and its negated inputs gives a total of four distinct transformations, Case d. and its negated outputs gives two distinct transformations



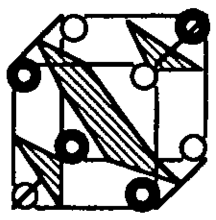


Fig 7.14 a. NOT(exclusive OR) separated with just two lines, b. Parity in three dimensions

For n≥ 5 the following induction argument on the number of input nodes. provides the proof. A proof for n= 4 is just the application of the proof procedure given below, taking into account the small number of nodes involved in the problem.

The Hypothesis

There exists a tight representation of the worst n dimensional transformation using n nodes, whose n-1 dimensional splits are tight requiring n-1 nodes.

For n= 3 the worst case by enumeration is the parity problem. The parallel representation of the parity problem satisfies the hypothesis. Appendix E2 shows a suitable proof of the case n= 4 based on the methods developed in appendix E1. Therefore the hypothesis is true for n= 3 and n= 4.

Induction hypothesis

If there exists a tight representation of the worst n-1 dimensional transformation using n-1 nodes, whose n-2 dimensional splits are tight requiring n-2 nodes, then there exists a tight representation of the worst n dimensional transformation using n nodes, whose n-1 dimensional splits are tight requiring n-1 nodes.

Assuming the hypothesis for n= k-1, we examine the case for n= k. If we could solve every transformation in k nodes or less then the hypothesis would be proved so now assume we have the worst possible transformation f say. We prove that this worst case does not violate the induction hypothesis.

Proof of induction hypothesis

Due to the assumptions there exists a k-1 representation of the two k-1 dimensional splits of the worst k dimensional transformation under consideration. These representations are tight and their k-2 dimensional splits exist and are themselves tight. We can construct full representations of the k dimensional problem that are tight in the k-1 dimensional splits by using each of the k-1 dimensional representations given above to represent each side of the split of the k dimensional transformation. This is done via the technique of amalgamating the lower dimensional representations, as discussed earlier. We choose a representation that is minimal, the minimal representation for the worst case is by definition tight. The amalgamation scheme in appendix E3 shows that given the above conditions the worst k dimensional transformation requires at least k and no more nodes to represent it fully.

Since we have that the original hypothesis is true for n= 3 and n= 4 it follows via the induction argument that it is true for n≥ 5.

Size limitations of general n-m-p nets

The results on neural network size and topology can be generalised to the multiple output case. (Fig 7.15).

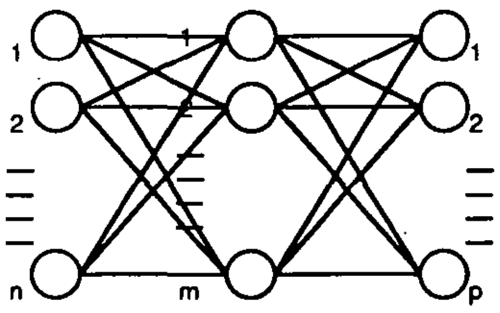


Fig 7.15 Multiple output neural network structure

Each of the p output nodes can be considered individually as single output problems and so we can get the quick generalisation that the n-m-p net can be solved as stated in the two layers of weights with a maximum of n*p nodes in the hidden layer. (See fig 7.16).

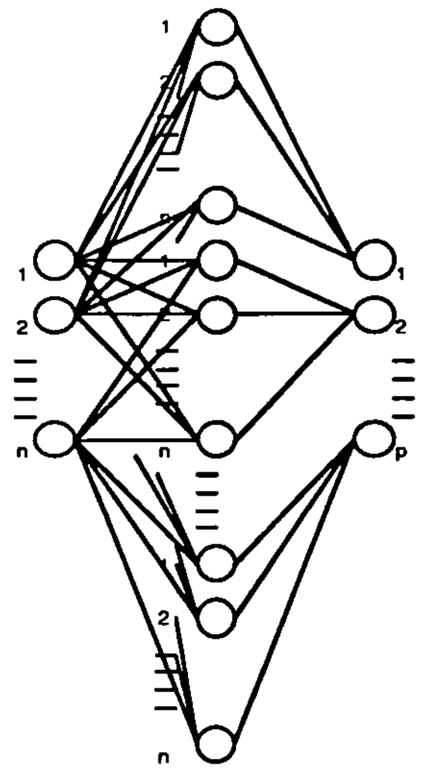


Fig 7.16 Three layer multiple output neural network structure in which the hidden layer consists of p distinct sets of n nodes

This net can be transformed into one of the form (n+ceiling($\log_2 p$))-m-1, where ceiling(x) is the smallest integer greater than or equal to x. This is carried out by considering pairs of output nodes as representing projections in a higher dimension. Doing this until we have only one output node results in the form as stated above and so this net can be solved with a hidden layer of m=(n+ceiling($\log_2 p$)). (See fig 7.17).

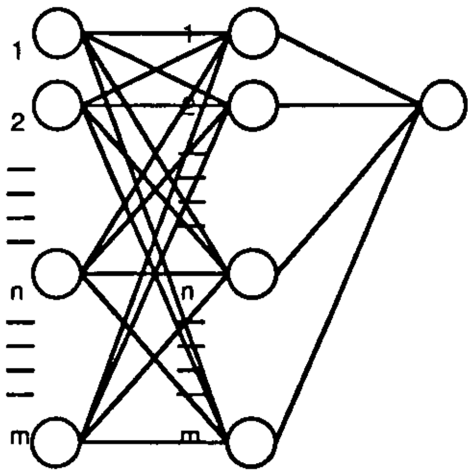


Fig 7.17 The multiple output neural network structure transformed to an equivalent single output neural network structure

A disadvantage with this form of the net is the loss of parallelism in execution. Namely the different output nodes must each be calculated separately by specifying the value of the ceiling($\log_2 p$) extra input nodes. A method that is successful in regaining the parallelism inherent in neural networks is transforming the net into the Loughborough form making use of the specialised m-type hidden nodes. The m-type nodes have a single set of weights and a given number (say k) of bias values which provide the (k) different output values. A single m-type node with k bias values behaves like k nodes with identical weight values. This is discussed further below.

The Loughborough net

To regain the parallelism of a general n-m-p net we remove the extra input nodes and use them to define the extra biases that must be applied to the m-type nodes.

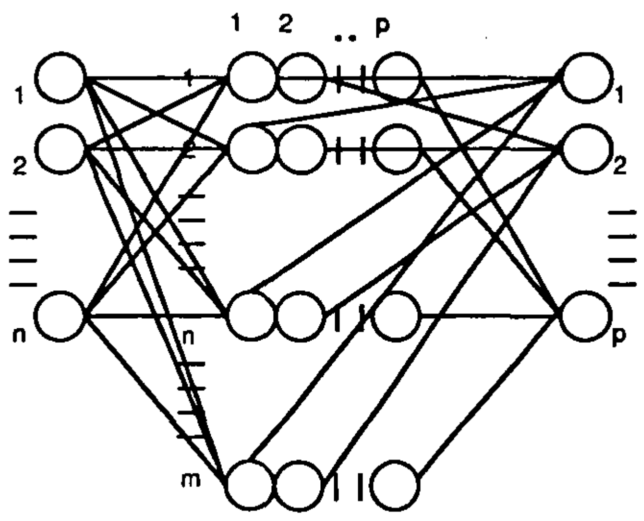


Fig 7.18 The Loughborough neural network structure. The single output network transformed to an equivalent multiple output neural network structure using m-type hidden nodes

The m-type nodes have ceiling($\log_2 p$) separate bias weights that are applied to the node. Each bias is applied after the node has worked out the input from the input nodes. These separate m-biases are then applied and the outputs passed on to the p output nodes.

The Loughborough form has the topology of an n-(n+ceiling($\log_2 p$))-p feed forward net with specialised m-type hidden nodes (fig 7.18). Therefore given any Boolean transformation, $f:B^n\text{-}>B^p$, in the Loughborough form requires a maximum of (n+ceiling($\log_2 p$)) m-type hidden nodes to model the problem.

Node Parallelisation

As demonstrated above, the size of network needed to model a problem can be greatly reduced if node parallelisation can be achieved. If we consider the parity problem, the planes that the nodes represent are all parallel, although there are in fact two classes of parallel plane of different polarity (see chapter three).

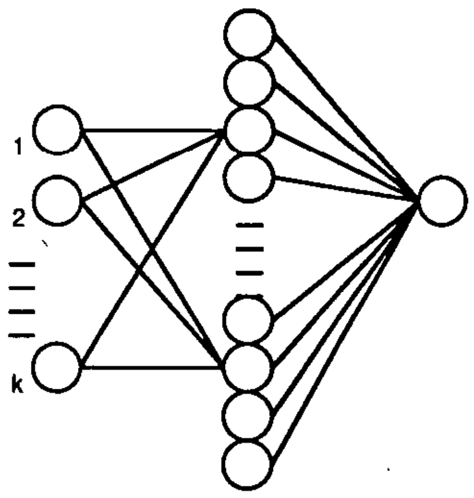


Fig 7.19 Single output neural network structure making use of singly outputting m-type nodes

This means that the parity problem could be modelled with two m-type hidden nodes for all dimensions. These m-type nodes in fact output to just one output node and so should be distinguished by classing then as singly outputting m-type nodes, or som-type nodes (fig 7.19).

Since we have successfully parallelised the parity case, the question arises as to whether in general we can make use of the property of parallelism in the node representation to reduce the problem. If we can spot parallel nodes in a representation, then we can make use of the som-type nodes. However this would be a very laborious search process and may in general not occur.

Summary

This chapter has given a new lowest upper bound on the number of nodes in the hidden layer of feedforward neural networks representing Boolean transformations with multiple inputs and a single output. This result was further generalised to multiple output cases. Specific transformations can often be modelled by fewer nodes than the limit presented in this chapter. Methods for producing this minimal representation require further study. Existing techniques include those of Tani et al('89), where an oversupplied network is pruned to obtain the minimal representation.

This chapter has examined techniques for amalgamating nodes in neural networks, so reducing the size of the neural network models. Transformations that can be implemented by sets of parallel hyperplanes in the hidden layer can be further reduced in size and complexity. This is achieved by amalgamating the parallel nodes and providing multiple bias weights to provide the function of the separate nodes.

This technique was introduced within the framework of the new neural network architecture, the Loughborough net. This architecture used two new node structures. These nodes consisted of multiple biases, forming the multiple outputting node, the m-type node and the singly outputting m-type node, the som-type node.

Chapter 8. Engineering Reliable Neural Systems

Outline of chapter

Previous chapters have examined neural network models of transformations from multidimensional Boolean input spaces to either single or multidimensional Boolean output spaces. This chapter examines transformations from real valued input spaces. The continuity of real valued spaces makes a great difference to the types of transformations that can be modelled. The results obtained in previous chapters can be transferred to the case of real valued input spaces if we consider training sets with discrete, isolated points whose output values are Boolean values.

The discrete nature of these input values allows us to construct atomic knowledge elements such as sandwich nodes which can then be manipulated and trained in much the same manner as the Boolean cases. The neural network size results do not generalise immediately since the topological structure of the two spaces are distinct. Several results by Mirchandani & Cao('88), Huang & Huang('91) and Baum('88) are applicable to the first layer of weights in the real input feedforward neural network. General results on network size and topology can be given by applying the Boolean network size results to the following layers of the network.

Continuous Spaces

The most significant property of real valued spaces is their continuity. Any transformation defined on such spaces containsan infinite number of data points. Training neural networks to model such transformations provide significant difficulties. Arai('89) discusses a scheme whereby an infinite three layer network can model arbitrary continuous transformations, but physical implementation of such systems would be impractical.

Transformations that are constructed from basic neural network atoms, that is linear summation nodes with linear or nonlinear threshold operators, can be modelled by neural systems. This is due to the algebraic identity of the transformations under consideration and the neural network model. This is demonstrated by the simple transformation, Z= 5X + Y + 10, whose neural model is shown in fig 8.1. The threshold function used in this network is just the simple linear identity threshold.

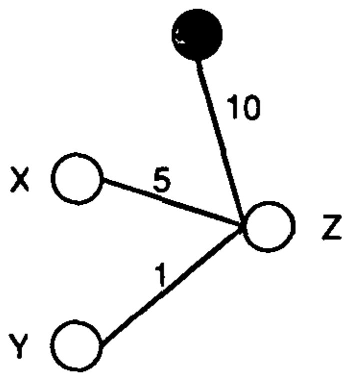


Fig 8.1 Neural network model of the transformation Z= 5X + Y + 10

Given an unknown transformation, the problem of finding a suitable neural network model becomes very difficult. If strong constraints are made on the size and architecture as well as the threshold functions of the network this problem is unsolvable in the most general case. Classifiers are studied by Baum('88), Mirchandani & Cao('88), and Huang & Huang('91). A different classification is made depending on the region of space that is under consideration. Feedforward neural networks with hard logical threshold nodes behave as classifiers and so are ideal for modelling classification transformations.

At the simplest level classifiers form a dichotomy on a set. This means that the neural network model requires only a single output node. The second simplification is an assumption on the structure of the real valued input spaces. The assumption is that the decision regions can be separated by rectilinear segmentation. This is required if we are to use the standard neurons with a hard limiting threshold. Several basic properties of these networks are dependent on the size of the network under consideration.

Table 1.2 (chapter one) shows the representational properties that depend on the number of layers of the network. Increasing the number of nodes in the layers of the network allows the implementation of more complex decision regions. To analyse such regions we must introduce some concepts relevant to rectilinear geometry.

A set of data points which can be separated by a single hyperplane is said to be linearly separable. A set of data points which can be separated by a finite intersection of hyperplanes is said to be separable by a single neural system.

Polytope classes

The intersection of hyperplanes has been studied by Mirchandani & Cao('88) and Huang & Huang('91). Huang & Huang has formalised the study, considering the hyperplanes as closed half spaces. The intersection of two or more of the spaces forms polytopes, which may be empty, bounded, or unbounded. A region formed by the intersection of two half planes is termed a two-polytope, three a three-polytope and n an n-polytope. Fig 8.2 shows several different polytopes, bounded and unbounded that can be constructed by the intersection of several half spaces.

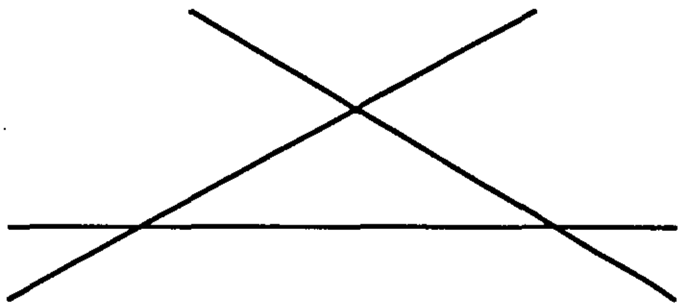


Fig 8.2 Three half spaces intersect to form six 1-polytopes, twelve 2-polytopes and six 3-polytopes

It is possible to gain some bounds on the number of nodes required to model the problem in question by examining the number of regions into which the input space is partitioned. Mirchandani & Cao showed that the maximum number of regions (M(H,d)) that are separable using H hidden nodes in a d dimensional space is given by the formula,

$M(H,d) = \Sigma^{d}_{k=0} (H,k)$, where (H,k) is the binomial coefficient such that (H,k) = 0 if H < k.

equation 8.1.

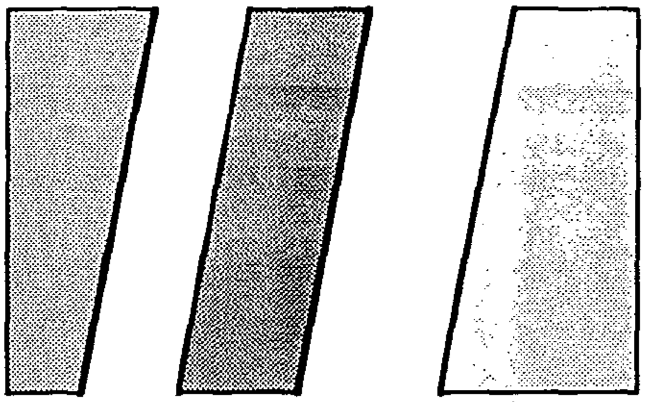


Fig 8.3 Five regions of the two dimensional input space require four parallel nodes to separate them

This result gives the minimum number of nodes required to model a particular problem. We find the minimum number H such that M(H,d) is greater than the given number of separable regions. The actual neural model of the problem in question may require many more nodes. At the simplest level, this occurs when we have parallelisation of the decision regions as discussed by Mirchandani & Cao. In fig 8.3 we have five decision regions requiring four hidden nodes when the formula would give M(4,2)= 11, that is the maximum number of decision regions in a two dimensional input space using four nodes is eleven. The case in question, the decision region parallelisation, is not a maximal case. Huang & Huang and Baum discuss more complicated examples where similar arguments apply, see fig 8.4a where seven data points require three nodes is a maximal case, while seven data points require four nodes to be separated in fig 8.4b.

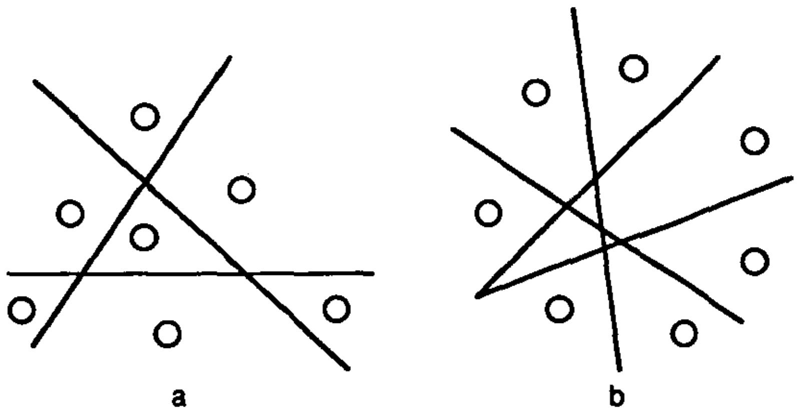


Fig 8.4 a. Maximal separation of seven points by three nodes, b. Non maximal separation of seven points requiring four nodes

The above analysis demonstrates that the number of nodes required in the hidden layer is highly dependent on the structure of the regions to be separated and not just the number of regions. The difficulty with finding methods of modelling such problems with neural networks is that once the structure of the classification regions are known, essentially the problem is solved. Referring to fig 8.5a, it can be seen that the boundaries of the separable regions represent nodes in the first layer of the neural network. The selection of the combination of suitable regions in classification occurs through two Boolean transformation layers (all Boolean transformations can be modelled by two layers of a feedforward neural network, see chapter seven) that follow the boundary decision layer of the network, fig 8.5b. Fig 8.6 shows the general neural network model of a transformation from a multiple real valued input space to a single Boolean output.

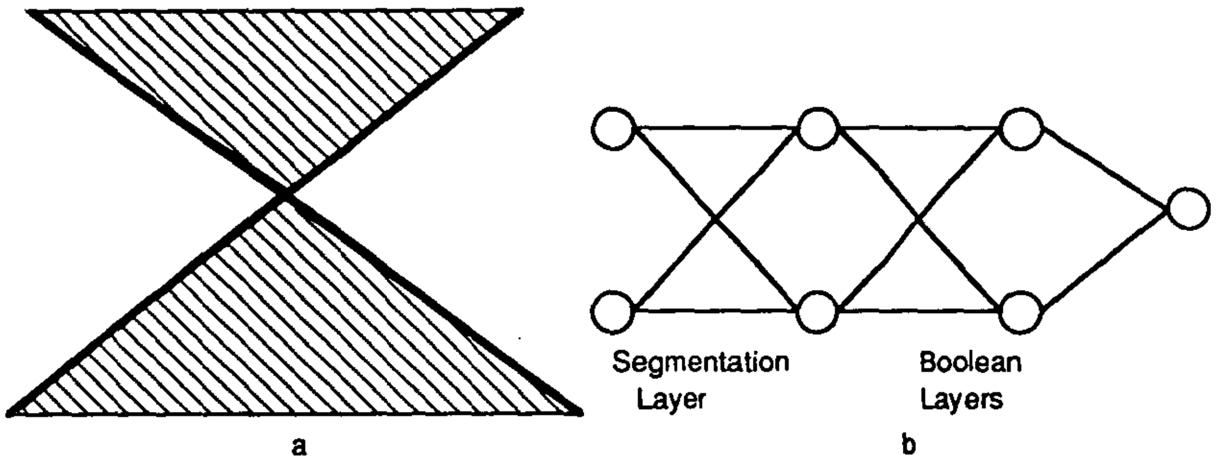


Fig 8.5 a. Four regions with an XOR classification transformation and, b. Its neural representation

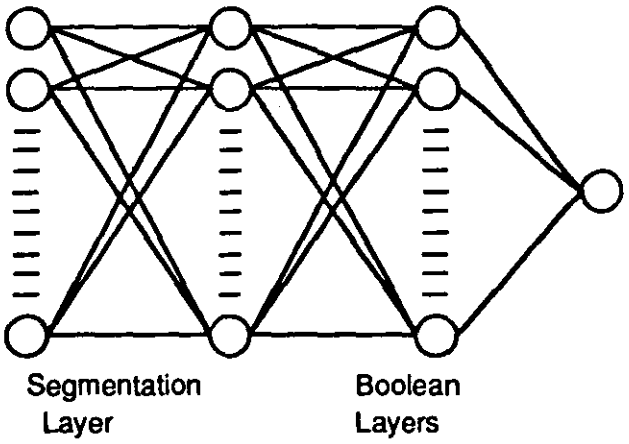


Fig 8.6 A general neural network model of a real valued classification transformation

Automated training

Many automated training regimes such as backpropagation make use of isolated data points. The classification of regions of the input space have no meaning except via the specification of the set of data points that make up the regions. Ideally for complete specification all the points in the continuous space of the input space must be specified. In reality this is impractical and so a finite subset of points specify the classification transformation. The points that are modelled by the node boundaries in fig 8.7 specify the XOR transformation shown in fig 8.5a.

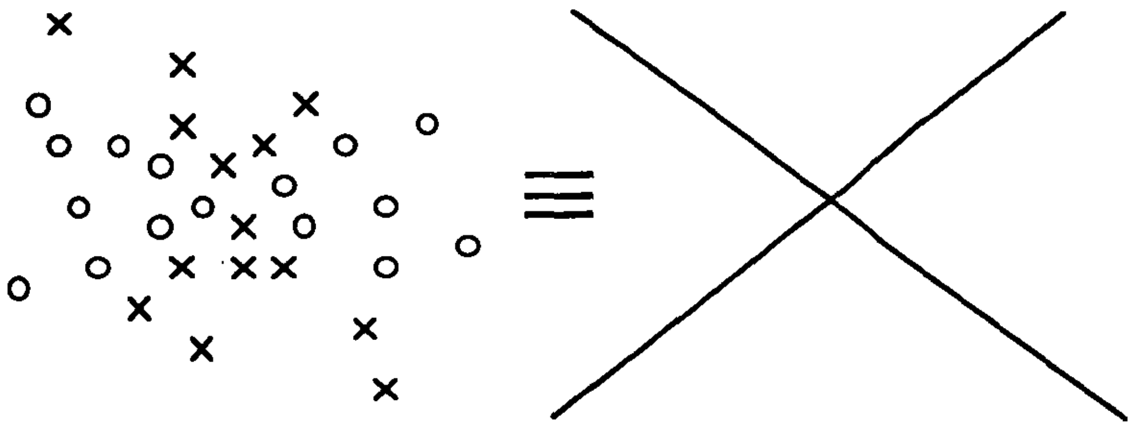


Fig 8.7 Separation of a real data set into XOR regions by two segmentation nodes

Given a finite training set with isolated training points, the notion of a unique regional classification is lost. The best possible segmentation of the data points can be sought, which would ideally represent the classification transformation being modelled. As discussed by Huang & Huang('91) and Arai('89), isolated data points provide the possibility of implementing arbitrary mappings on a training set. This is the case since each isolated data point can be associated with a unique output which can be suitably implemented by the manipulation of the weights in the network.

Properties of isolated data points in real valued spaces

Interleaved rectilinear points from different classes must be separated by multiple interleaved planes. The separating planes are linear in structure and so a single plane (or a reduced number) can not separate the points in question. Fig 8.8a shows five rectilinear points that must be separated by four nodes. Anything less, for example a single node can not separate the whole set (fig 8.8b).

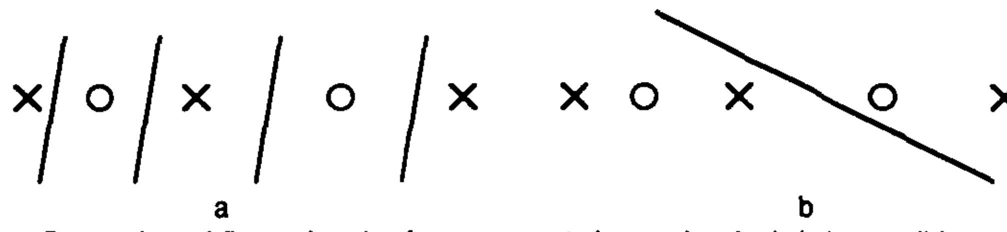


Fig 8.8 a. Separation of five points by four segmentation nodes, b. It is impossible to separate the points with fewer nodes

A pair of nodes can isolate n points in an n dimensional input space. If a point from a different classification lies on a line between any two such points or on the plane intersecting the n points, more nodes must be added (as above) to fully separate the set. Fig 8.8c shows how two points in two dimensions can be separated by two nodes. These do not interfere with any other data points. In fig 8.8d the point from another class lies within the segmentation and so more nodes are required to fully separate this set.

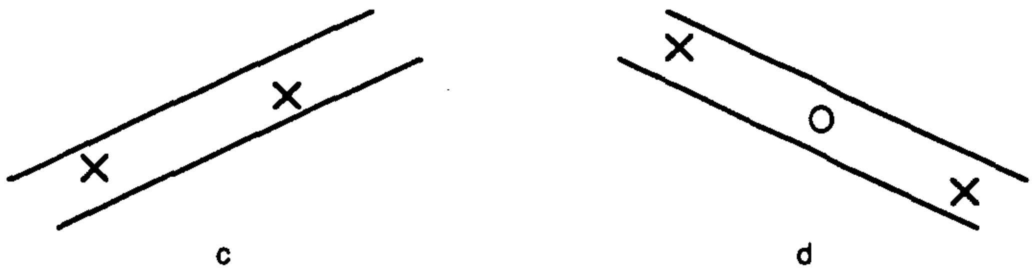


Fig 8.8 c. Two rectilinear points can be separated from the rest of the input space with a pair of segmentation nodes, d. Three rectilinear points can not be separated if the points are from different classes

Baum('88) provides some result on the number of hidden nodes required to model problems in which the data points are in general position. Data points in an n dimensional space are in general position if there are no subsets of these data points, with n + 1 elements, lie on an n -1 dimensional hyperplane. Baum demonstrates that an N element set in a d dimensional space can be separated by a ceiling(N/d) nodes into arbitrary dichotomies. Fig 8.9 shows an example in two dimensions that requires this limit to model the problem.

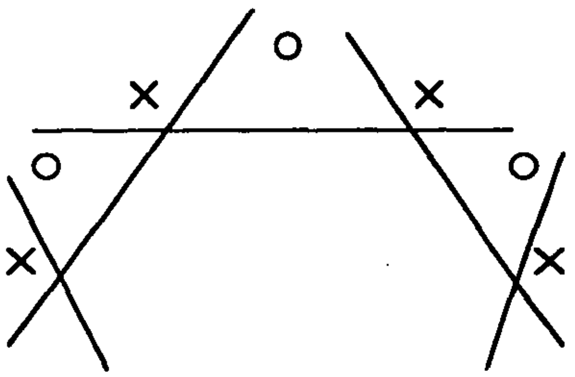


Fig 8.9 Separation of N points in general position on a circle by ceiling(N/2) segmentation nodes

General classification

If we now consider general classification problems which have multiple output classes, the analysis becomes more complex. Making the assumption that each point must be separated from every other point in the training set provides an upper bound on the number of nodes required to model the training set. These assumptions mean that M-1 nodes are required to separate M points even if all the points are not rectilinear. This result is quoted by Kung & Hwang('88), Mirchandani & Cao('88), Baum('88) and Arai('89). The results of Mirchandani & Cao can be extended to this case. The number of points become the number of decision regions that we have. Similar problems as discussed earlier apply to this case if the separation scheme is not maximal. The formula does not provide the correct maximum number of nodes required to model nonmaximal cases such as rectilinear data points.

Padalines

Given classification problems defined on isolated data points, the accuracy of the neural network model will depend on a number of factors. The first is the number of nodes in the network. The greater the number of nodes available the greater the accuracy of the neural model of the transformation in question. This is demonstrated by the fact that the four node neural network can model the classification problem of fig 8.8a while a single node network can not. The greater the number of nodes available the greater the number of neural models that can represent the classification problem.

The other major factor that determines neural network model accuracy is the the type of nodes that we use. Polynomial adalines were introduced by Sprecht('67) and Hinde('74). They allow input parameters to be transformed polynomially, so removing the constraint that a node boundary has to be linear in the input space. Caudhill('88) provides an introduction to the algorithm and its uses. The representational power of neural networks can be altered by applying various different transformation to the input layer. Sprecht's('90) statistical functions and fuzzy neuronal systems such as, that make use of B-spline maps or fuzzy masks and the CMAC technology (Kraft '90) all improve the representional power of neural networks by providing different input transformations to the network.

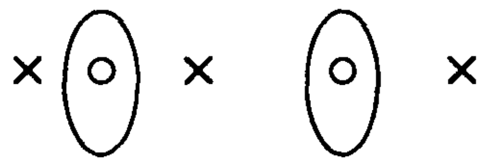


Fig 8.10 Separation of five rectilinear points using two polynomial segmentation nodes

Fig 8.10 illustrates how the five rectilinear points of fig 8.8 a- d can be separated with two padaline nodes. The power of these input transformations can also be illustrated by the example shown in fig 8.11a. By applying a squaring coordinate transformation, the circular region shown can be implemented in a single layer with the inputs, X, Y, X*Y, X^2, Y^2. The corresponding standard neural model (8.11b) would require many more nodes to isolate region and require more hidden layers to implement the transformation required to model the problem.

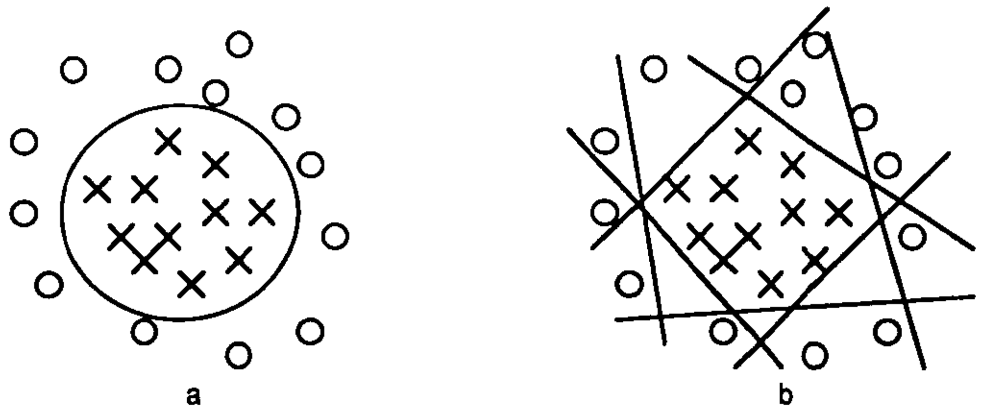


Fig 8.11 a. Polynomial separation of the training points requiring a single segmentation node, b. Linear separation of the data points requiring many segmentation nodes

Finding the relevant transformation for problem in question becomes an automated learning task. All the possible polynomial transformations, that is first order, second order, etc, up to the model required, are provided as inputs to the network. The coefficients associated with this transformation are learned as the first layer of weights in the network. Therefore irrelevant input transformations have weights that decay to zero, while relevant input nodes are learned via the back propagation of the error. This means that we can provide a padaline neural model of a transformation to any given order. The learning algorithm will learn the shapes of the decision algorithm and identify the relevant transformation involved.

Introducing other input variable transformations allow different properties of a training set to be modelled. Periodic and exponential functional transformations of the input variable allow relevant physical systems to be modelled by neural network systems.

Learning input quantisation values

Given linear summation units and hard limiting threshold elements, the boundaries of the decision regions are straight lines and the output after the first hidden layer are Boolean values. Therefore the first layer can be considered to be a quantisation layer. Real values are input to the hidden layer and quantised Boolean values are output. These Boolean values are then manipulated by the neural network structure providing suitable output values.

The behaviour of the training algorithm over various different input sets given different neural network structures are examined in appendix F1 and F5. The real input quantisation is examined and the ability of single layers of nodes to model arbitrary quantisations discussed.

Real output nodes

Dealing with transformations that provide real output represent a significant difficulty. The data passing between the internal layers of a neural network are essentially Boolean in characteristic, even if the values are actually real values near +1 and -1. This is due to the shape of hard limiting thresholds. All the output values of the internal nodes tend towards either +1 or -1 preserving the Boolean structure.

Constructing real valued outputs from these Boolean values requires a decoding layer of weights. The magnitude of real valued outputs is provided by a suitable multiplication of the quantised values by the weight values. The output is in fact a quantised real value and not a continuous real valued output.

Providing more output quantisation nodes allows a finer grain of coding. The problem associated with output decoding is that the quantisation values can not be easily learned. Ideally we would be able to train the values of the output decoding layer, such that the best decoding scheme is used to model the transformation under consideration. Appendix F2 & F3 discusses experiments that investigate the learning properties of neural network when modelling transformations from Boolean inputs to a real valued output.

Real inputs and Real outputs

Neural network models with real input and real output nodes have a quantisation layer of weights and a decoding layer of weights as discussed above. A Boolean transformation must be implemented between these two structures. The results of chapter seven show that two layers of weights are required to model Boolean transformations. Therefore models of real input real output transformations will consist of four layers of weights.

Appendix F4 discusses some experiments for training neural networks to model transformations from real valued input spaces to real valued output spaces.

Network size and topology

The results on network size and topology presented in chapter seven can be generalised to neural networks with real valued inputs. The structure of these networks is that of a quantisation layer followed by the Boolean network structure. Therefore if the size of the quantisation layer is known the network size results immediately apply.

Given an n dimensional real input space with D linearly separable regions, the work of Mirchandani et al provides a limit on H the number of nodes in the first layer, see equation 8.1. M(H,d) is the maximum number of regions that can be separated by H nodes in a d dimensional space. Therefore given a dichotic classification problem with M(H,d) linearly separable regions that are maximally separated by H nodes in the quantisation layer, we require another H nodes in the hidden layer of the Boolean layers (given by the result in chapter seven). If we do not have a dichotic classification, but a transformation to p output nodes, we require H + ceiling($\log_2 p$) hidden nodes in the Boolean transformation layers. This case generalises the Loughborough network to real valued inputs and uses m-nodes in the hidden Boolean layer of the network.

If we have D data points in general position in a d dimensional space, as discussed by Baum, then we derive the result that arbitrary dichotomies on the set can be modelled by networks with a ceiling(N/d) nodes in the first hidden layer(see Baum '88) and similarly ceiling(N/d) in the second hidden layer (see chapter seven). Since the transformation forms a dichotomy only a single output node is required.

Methodology for designing reliable neural systems

The design methodology places the implementation of neural systems into the area of reliable systems. Each stage of the design process is well understood and the behaviour of the network explained in terms of the atomic nodes that contribute to the decision processes.

The system identification phase investigates the structure of the input space identifying the significant components of the decision algorithm. The quantisation layer fine tunes these structural identifications providing the most optimal quantisation value via the trained weights. The Boolean layer of the network implements the best transformation that can model the given data set. Essentially it identifies the knowledge structure underlying the data. The atomic knowledge elements provide a succinct representation of this knowledge that could be transformed to a disjunctive encapsulated rule set.

The total neural network structure is reliable since its behaviour is well defined and can be predicted. Not only does the network model the given training set, but it has predictable generalisation properties. The generalisation properties are predictable since the atomic knowledge elements do not interfere in an unknown manner. Everything about the network is well known. The nodes and knowledge elements possess the desirable properties of atomicity.

A neural network is designed to model a specific set of data by constructing a suitable network structure of nodes and to specifying the weight values of the connections between the nodes. Automated search techniques exist for finding suitable network structures, (Mirchandani et al). However they involve a high computational load and are not guaranteed to produce optimal neural network models.

The work of the previous chapters have allowed the behaviour and structure of neural network systems to be understood and explained. This in turn allows a general design methodology to proposed for the construction of reliable neural network systems. The structure of the designed system will be that of a feedforward net with the following components;

i. Input transformations,

ii. Input quantisation layer,

iii. Boolean transformation layers,

iv. Decoding layer.

These components are illustrated by the network structure shown in fig 8.12. The transformed input values are fed to the input nodes of the neural network.

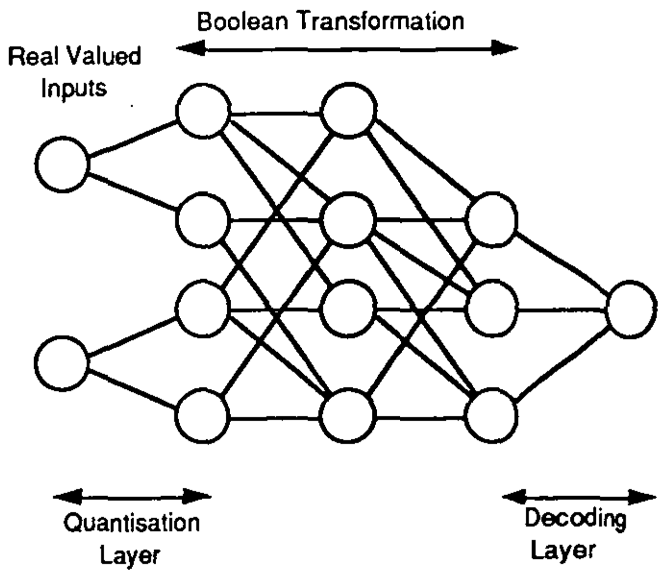


Fig 8.12 Real input real output neural network structure

System Identification

The set of training data is presented which must be modelled by a neural network. If no specific output nodes are designated then the dependent output node in the data must be identified. The conditions that must be satisfied are that all the output nodes are dependent variables of the input nodes. That is, each unique input datum is associated with only one output value, the transformation is an injective function.

The nodes in a neural network only provide linear combinations of input nodes that are then thresholded. Therefore the separated regions of the input space will only have linear boundaries. This can be overcome with the use of input variable transformations, for example polynomial transformations. The nonlinear relationships that exist between the input variables must be identified. Decoupled models of the system provides a suitable method. For a fixed output value, the decoupled variables are correlated by various polynomial relationships. The polynomial transformations identified define the shape of the boundaries of the quantised regions in the input space.

Quantisation layer

Having identified the system under study the quantisation layer specifies the position of the boundaries of the segmented regions of the input space. This is achieved via the variation of the weights in the quantisation layer. The quantisation layer defines the number of different quantised regions of the input space since the interaction of the different regional boundaries of the nodes occurs here. Appendix F1 and F5 describe experiments that investigate the ability of a single layer of quantisation nodes to model arbitrary transformations from real to Boolean spaces.

Boolean layer

The Boolean transformation layer provides the actual functional transformation between the input quantised regions and the output quantisation units. If the relevant transformation is well known, then it can be immediately implemented with the relevant neural network nodes. If only the input/ output data is known then the layers are trained automatically via the backpropagation technique. Appendix F3 and F7 describe experiments that investigate the ability of a multilayer of nodes to model arbitrary transformations from real to Boolean spaces. Effectively this is a system of a quantisation layer followed by two Boolean layers of weights. This system is more able to model the arbitrary transformations than the single layer quantisation system since the Boolean layers can model arbitrary transformations between the input and output layers.

Decoding layer

The output quantisation represents the acuity that is required of the transformation.

This is related to the input/ output data that the system is required to model. If there is greater variety in the output signals the larger the number of nodes required to feed into the decoding layer. This follows from the fact that the greater the number of possible signals, the greater the acuity that can be achieved by the decoding nodes. Appendix F2 and F6 describe experiments that investigate the ability of a single layer of decoding nodes to model arbitrary transformations from Boolean to real spaces.

This design methodology has been applied to the design of a neural network controller. This is discussed in chapter nine. The design process can be very laborious as the identification of the relevant system variables is a difficult task. The design methodology structures the implementation of the neural network systems so that their structure, function and behaviour are well understood.

Summary

This chapter has discussed neural network models of real valued transformations. Real valued transformations that are defined on a finite set of discrete training points were modelled by neural networks systems.

Real valued neural networks were shown to consist of four layers of weights, the quantisation layer, the two Boolean transformation layers and the decoding layer. These insights led to the generalisation of the results on neural network size and topology to the real valued cases.

Finally a design methodology was described which allowed reliable neural network systems to be implemented.

Chapter 9. Controlling Processes with Neural Networks

Introduction

This chapter addresses the design and implementation of a neural network controller for the dispensing of adhesives in the manufacture of mixed technology printed circuit boards (Chandraker et al '89 and Barbiarz '89). Adhesive dispensing is one example of the many industrial processes (metal cutting, Wright et al '91, arc welding, Karsai et al '92, cement manufacture, Haspel '86 and other batch chemical processes) that are not amenable to standard control techniques. The standard techniques are not suitable since they can not deal with unpredictable process variations and process faults that may occur (Williams '90 and Antsaklis '92). This is discussed further below. Table 9.1 shows other manufacturing processes that have required intelligent control techniques such as expert systems, fuzzy controllers and neural network controllers.

Neural Networks have been used to model control processes (Barto '90), that is aid in the system identification phase of control process design, control process optimisation (Barto '83) and the actual control of processes. This chapter deals with neural network techniques that control processes (Miller at al '90 and Grant et al '89).

A control problem typically consists of the maintaining the process outputs between set limits. This is achieved by identifying the actions that must be taken when the process approaches or exceeds these limits. These limits are reached either because of process drift or catastrophic process errors.

Hard control problems that can not be solved by current techniques are generally only partially understood. A complete mathematical model of the process does not exist or cannot be found. Some of the process characteristics can be described and modelled mathematically, whereas others can not. To date, ad hoc approaches such as trial and error implementation of feedback control have been used. Since no formal models of these controllers exist robustness and stability measures cannot be easily provided. A systematic approach must be developed so that each control process can be understood. This will allow stability and reliability to be ensured.

The intelligent controllers of the future must be able to recognise and predict the onset of the catastrophes in order that suitable corrective measures can be applied. Control systems that can be partially designed and then trained automatically are of great importance in hard control processes. Neural network learning offers a method for automatic training that is very attractive. However, they are difficult to configure and interpret after training (Materna et al '89, Hertz et al '91 and Mirchandani et al '89). Interpretation of the neural network representations is extremely important, especially in safety critical applications. The structured neural networks discussed in this chapter allow a readily interpretable control model which is a great advantage over standard neural approaches.

Process	Control Technique	Reference
Blast Furnace	Fuzzy Controller	Sakurai et al '89
Adhesive Dispensing	Expert Controller	Chandraker et al '90
Multi-layer Bottle Blowing	Expert Controller	Morgan '90
Cement Kiln	Fuzzy Controller	Efstathiou '85 & '89
Metal Cutting	Linguistic Controller	Bourne '86 & '87
Grinding	Fuzzy Controller	Pei Yan et al '90
Lumber Cutting	Expert System	Massey et al '90
Wafer Etching	TI Etch Diagnostic System	Budge et al '90
Newsprint Production	Expert System	Opdahl '89
Fermentation	Expert/ Neural Estimator	Aynsley et al '89
Gas Arc Welding	Neural Network	Karsai et al '92

Table 9.1 Manufacturing processes that have been controlled by intelligent techniques

It will be shown that many functions relevant to control processes, such as system variable thresholding and system variable segmentation into acceptable and unacceptable bands, can be implemented with simple neural networks. The linear threshold properties of nodes allow for immediate quantisation of the system variables, while the Boolean transformation layers allow intelligent control actions to be constructed. The process failures can also be monitored by neural network structures. These provide suitable corrective actions when the failures occur. The structured combination of these neural networks represents the design of the total control process. Parts of this chapter have been presented in Messom et al '92.

Control Processes

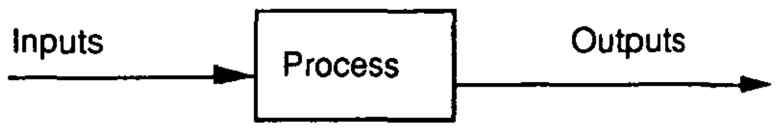


Fig 9.1 a. Schematic representation of a process transformation

A process (or plant) can be characterised by the input output transformation of the state variables of the system over time (see fig 9.1a). The schematic representation of open loop control is shown in fig 9.1b in which the corrective control actions are applied to the process inputs. Such a system is prone to steady state errors in process performance as well as being susceptible to process drift. Feedforward control requires a very good model of the process in order that the performance of the system is robust and reliable.

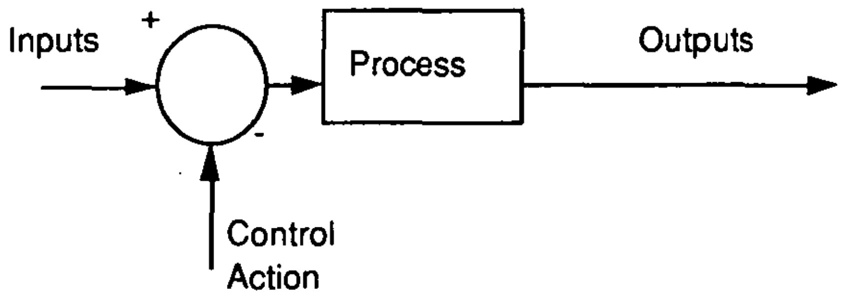


Fig 9.1b. Schematic representation of a feedforward control process

If feedback is applied to the system the performance of the process can be constantly monitored and the control actions adjusted. Fig 9.1c shows a schematic representation of the feedback control system. This means that steady state errors are minimised and the process drift due to external factors reduced. In practise problems exist in this system due to the variable or unknown time lag between the process action and the application of the control action. In the worst case the time lag can lead to undesirable oscillatory behaviour of the system. The tools of standard control attempts to provide control systems without these undesirable properties.

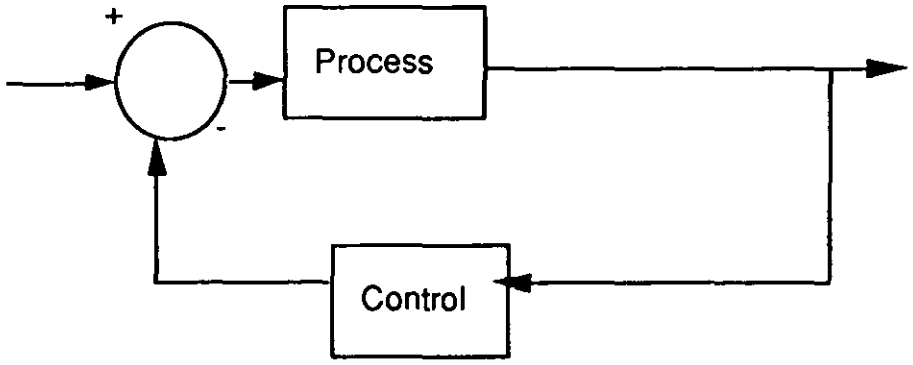


Fig 9.1 c. Schematic representation of a feedback control system

Adhesive dispensing

The neural network techniques discussed in the previous chapters is applied to the design of a controller adhesive dispensing. The specifics of the application will be briefly discussed while the general properties of neural networks in control systems will be discussed in depth.

The process investigated in this study is the dispensing of adhesive used in the manufacture of mixed technology printed circuit boards. The adhesive is dispensed on the printed circuit board to secure the surface mount components prior to wave soldering. Several techniques for dispensing the adhesive exist, pin transfer, screen printing and pressure syringe dispensing. Bridgeman ('89) and Barbiarz ('89) discuss the advantages and disadvantages of the various techniques. Problems associated with the screen printing include smudging of adhesive, while the pin transfer method can only apply equal sized adhesive blobs. Both systems suffer from the fact that they can only be applied to flat surfaces but have the advantage that adhesive is applied quickly and simultaneously over the whole board. The pressure syringe method can only dispense individual blobs but can perform well on any surface. The pressure syringe method also offers great flexibility in blob size dispensed and does not rely on guaranteed adhesive homogeneity if it can be counteracted by suitable control process applied to the system.

The system studied consisted of the following components.

i) dispensing unit;

consisting of a syringe of adhesive coupled to a pressure control unit.

ii) Seiko RT3000 robot manipulator;

that positioned the dispensing unit and the image processing unit at the appropriate place on the board.

iii) Image processing system for visual feedback;

consisting of Imaging Technology ITI 151 coupled to a camera with a magnifying optical system.

iv) The control system;

The structure of the system is shown in fig 9.2 (Chandraker et al '89, West et al '88).

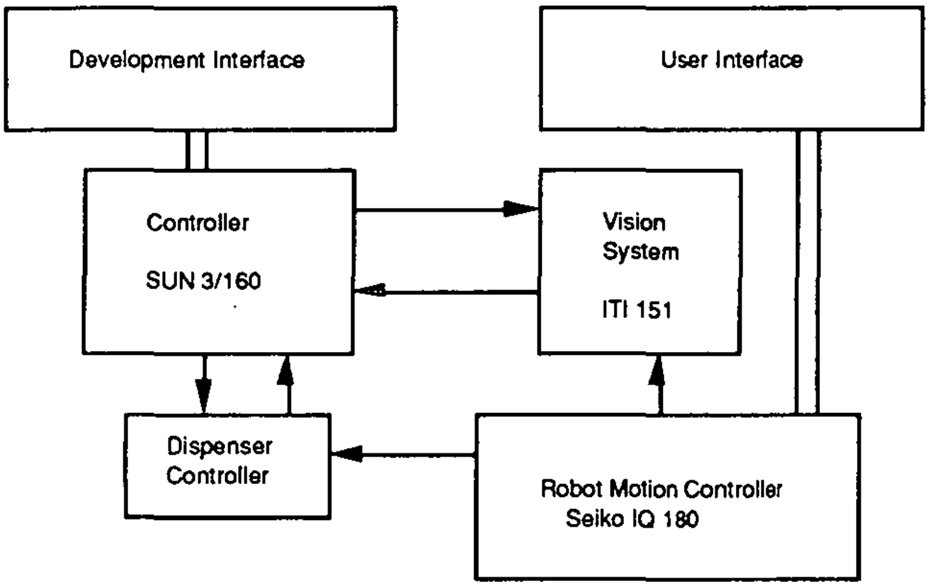


Fig 9.2 Structure of the controlled adhesive dispensing system

Process Characteristics

The process studied requires specified volumes of adhesive to be dispensed at well defined positions on the printed circuit board. A syringe contains the adhesive which is dispensed via the application of a pressure pulse to the adhesive. The volume of the adhesive dispensed is related to the properties of the adhesive and the pressure pulse applied to the adhesive. Variations in the adhesive include temperature sensitive viscosity, erratic thixotropic behaviour (that is time variant viscosity due to the application of shear forces) and material inhomogeneity.

A model for the process cannot be derived since the relevant process variables are not known and those that may be relevant (such as syringe nozzle pressure, velocity and viscosity variations) cannot be measured by available technology. The properties of the adhesive dispensing process are discussed further below, highlighting the properties that make the derivation of a suitable mathematical model impossible.

The control variables associated with the adhesive dispensing process are the pressure pulse characteristics. The pressure pulse is defined by the pulse height and width, the rise time and the fall time. Fig 9.3 illustrates the trace of the pressure pulse. Varying the pressure pulse height and keeping the pulse width constant effectively simplifies the manipulation of the control parameters, reducing the dimensionallity of the system. Keeping the pulse width constant also simplifies any timing constraints associated with the process. The pulse width can be increased in the exceptional circumstances that a large pulse area is required which cannot be achieved by the maximum available pulse height. The control model of the system can be simplified by decoupling the system into smaller noninteracting components. Essentially the controller becomes modular in design.

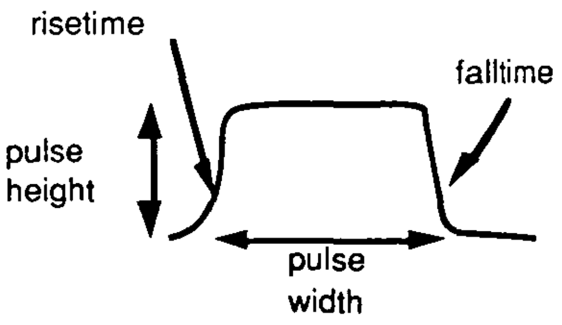
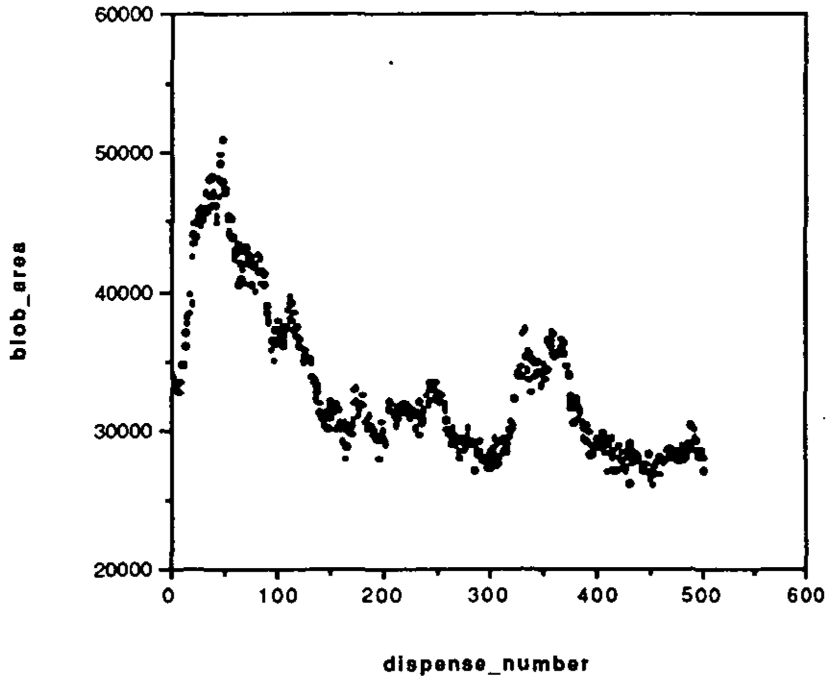
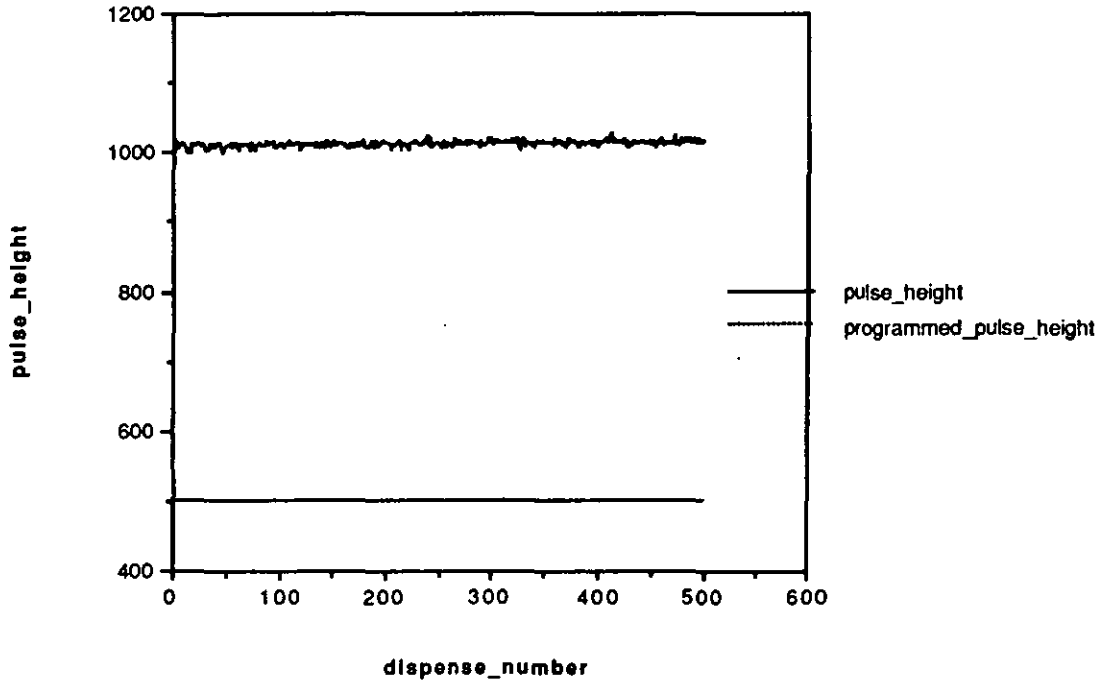


Fig 9.3 The pressure pulse variation within the syringe

Graphs 9.1a and 9.4a shows the variation in blob area (the output performance) for a fixed open loop input signal (fig 9.1b). There is extensive variation in output performance, the area of the blob produced varies greatly and in an unpredictable manner, for the fixed pressure pulse applied to the system. There are low and high frequency components in the variation. Some physical insight can give an explanation for this variation. Material inhomogeneity may explain the low frequency variation. The high frequency variations are due to the onset of bubbles in the flow (as discussed below). Variation in external temperature and pressure also effects the system, since a 10 °C variation in temperature can halve or double the viscosity of the adhesive (Bridgeman '89). The blob size dispensed for a given pressure pulse is related to the amount of adhesive that remains in the syringe and with improvements to the dispensing system, variations of over 7% have been observed between full and empty syringes (Knapp '87).



Graph 9.1 a. Open loop performance of the adhesive dispensing system



Graph 9.1 b. The pressure pulse variation that was applied to the system

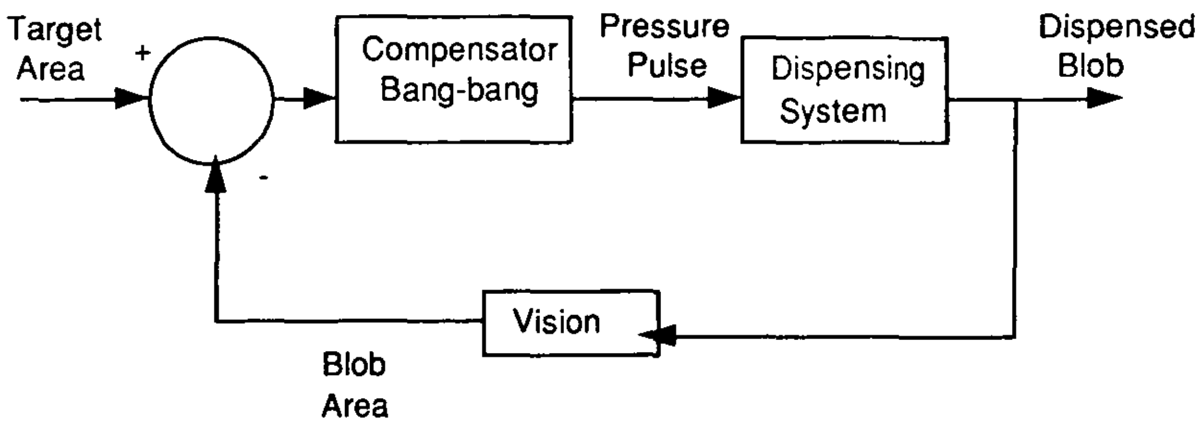
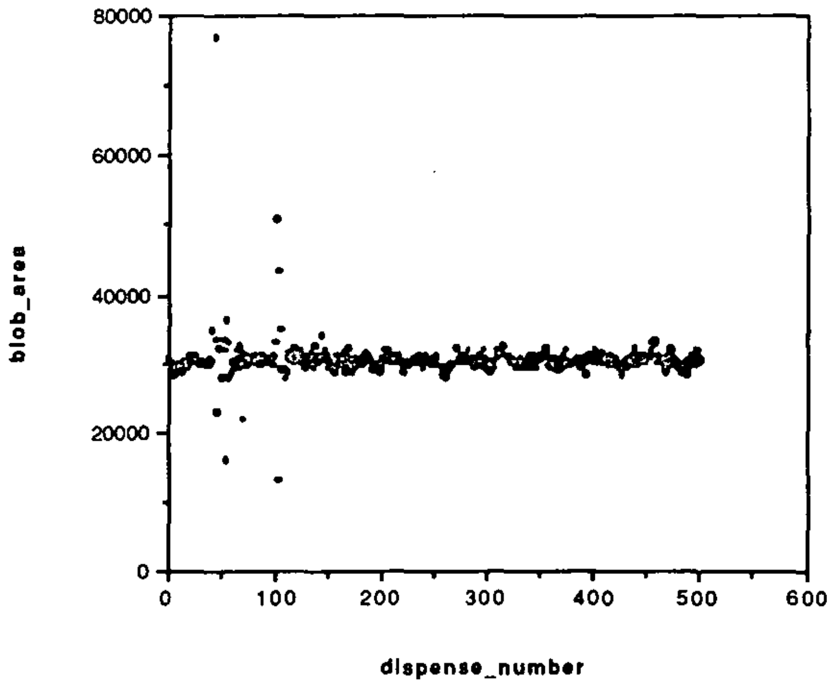
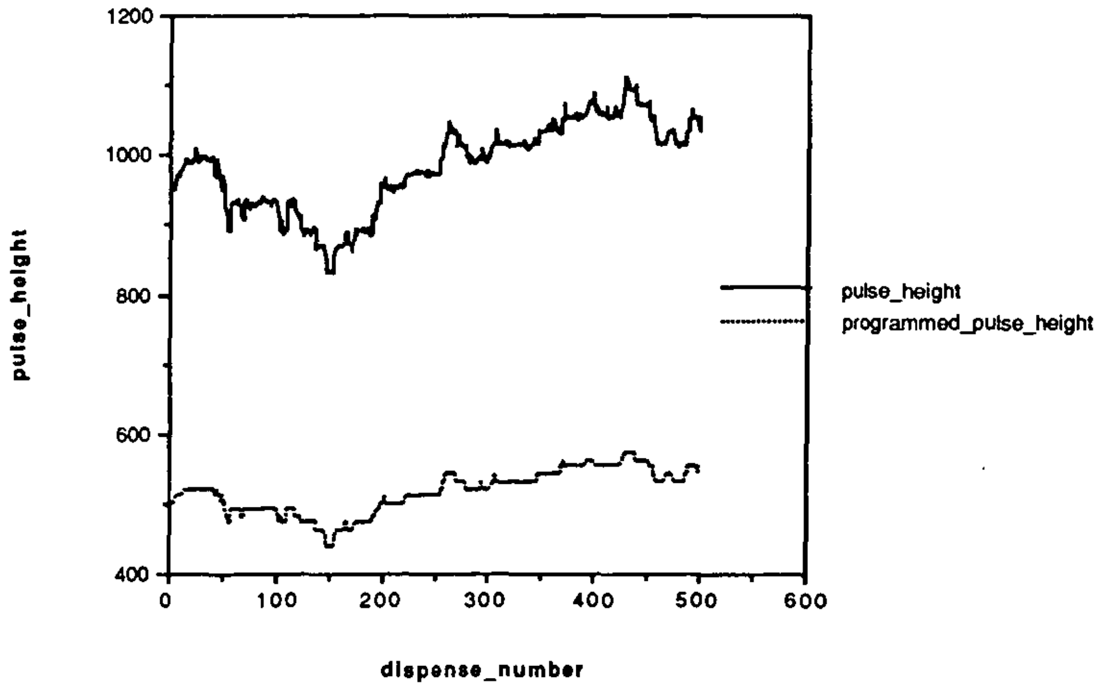


Fig 9.4 Feedback control loop of the adhesive dispensing system

The system was run closed loop using a rule based controller that exercised the full range of the pressure variable while maintaining satisfactory control although the process failures due to the existence of bubbles still occurs. Graph 9.2a shows the performance of the output variable and Graph 9.2b the pressure variation required to produce this performance. Photograph 9.1 shows the dispensed blobs during an experiment that was under control. Correlation analysis between the pressure variation and the blob area performance criterion show that there was no significant correlation between the two variables. The variation in the system is essentially stochastic and not correlated with the measured system variables (West '92). The variation therefore can not be predicted and makes this example a hard control problem. The open loop process variations are small from dispense to dispense and so the system can be controlled.

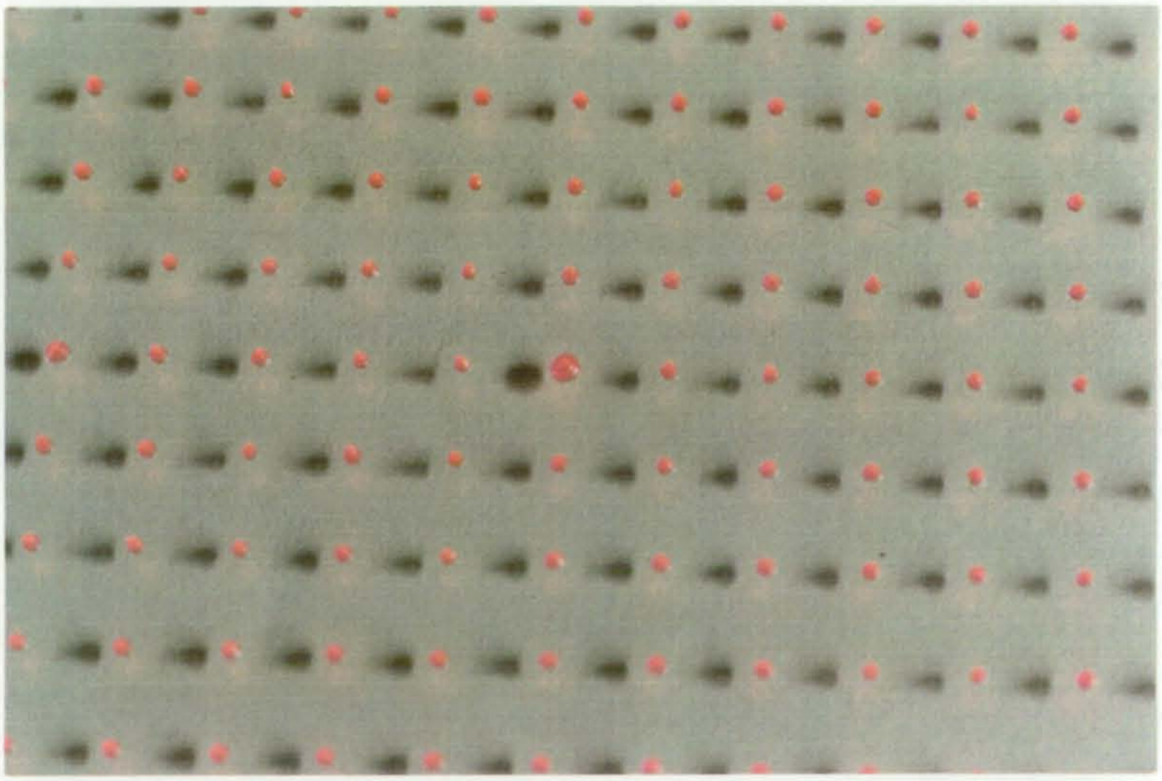


Graph 9.2 a. Closed loop performance of the adhesive dispensing system



Graph 9.2 b. The pressure pulse variation that was applied to the system

Mathematical models of specific runs or sets of runs can be constructed, but these models do not provide a predictive model of subsequent runs. Any particular run can be modelled by a polynomial of required order, but this does not represent a general model of the system. A simulation model of the system can be constructed which possesses the qualitative properties of the real system. That is low frequency variation in output performance, high frequency stochastic noise and the onset of process failures in stochastic time. The various catastrophic process failures that can occur in the dispensing of adhesive and the methods for dealing with them using neural networks are discussed below.

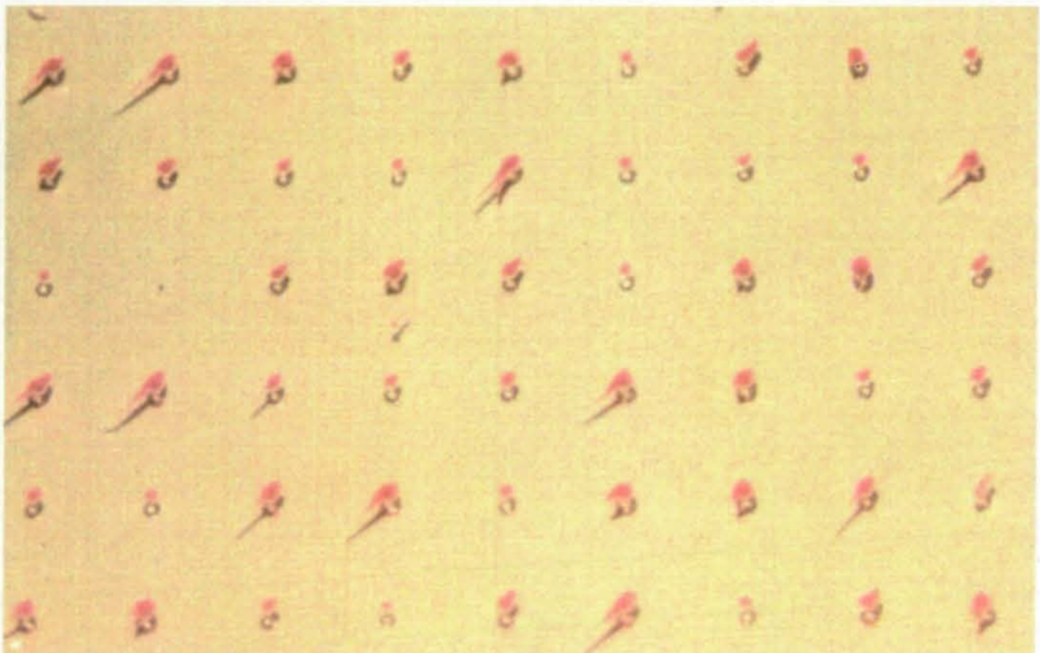


Photograph 9.1 Dispensed blobs during an experiment that was under control. Note the general uniform blob size. Note the excess adhesive in the centre of the photograph which is a characteristic precursor to a void

Process Faults

The controlled run of Graph 9.2a shows that in this case the bang-bang controller was very successful. The output area is maintained between the 10% error bands for most of the dispenses. The simple bang-bang controller is unable to deal with the onset of failures due to process faults. The process faults associated with the adhesive dispensing process are the onset of voids in the adhesive leading to erratic blob size, soiling of the solder pads due to the dragging of blobs and the sticky solenoid valve problem which leads to the soiling of the printed circuit board with the excess adhesive dispensed and the airline loading which leads to drift in the pulse height produced so effecting the blob area output performance.

The volume of the blob of adhesive dispensed is the important criterion for determining if a particular dispense has been good or bad. The second criterion is that it is well centred and in the position required. The position of the blob is well defined via the robotic system and performs well to the given tolerances (West '88). The volume of the blob can be determined from the plan area and height of the blob. For a good dispense, that is a circular well centred blob, the height is fairly uniform (Knapp '87) and so the main feedback variable is the plan area of the blob dispensed.



Photograph 9.2 Process faults due to the blobs that have fallen over. The tails of the blobs are likely to contaminate the adjacent solder pads

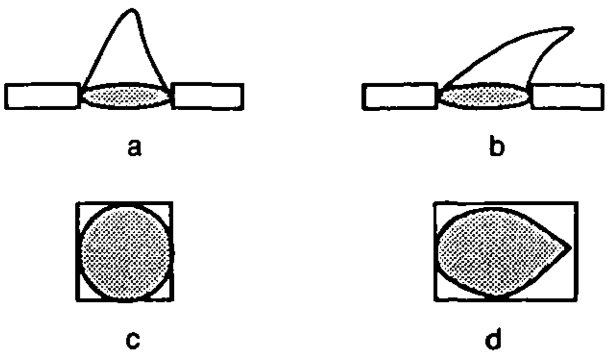


Fig 9.5 a. A good dispense, b. A bad dispense, c. Plan area and circumscribing box of a good dispense, d. Plan area and circumscribing box of a bad dispense

A good dispensed blob forms a conical shape, fig 9.5a. When the syringe is moved to the next dispensed position the blob may be dragged, fig 9.5b. This produces the process fault of the blob falling over which can soil the solder pads on the printed circuit board. This process fault is heightened by the variation in the stringy nature of the adhesive. The stringier the adhesive the greater the likelihood of the blob being dragged. Photograph 9.2 shows blobs that have been dragged. The box area ratio, that is the ratio of the plan area of the dispensed blob and the area of its circumscribing box, proves to be a good measure of a good dispense. A circular blob (a good dispense, fig 9.5c) will achieve the ratio of approximately 0.78, while a blob that has been dragged (a bad dispense, fig 9.5d) will be below this threshold value, fig 9.6. Graph 9.3a shows process data in which we have a good box area ratio performance, while graph 9.3b shows a bad box area ratio performance.

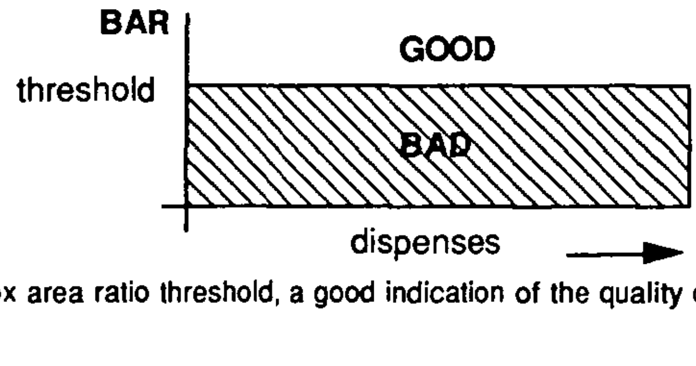
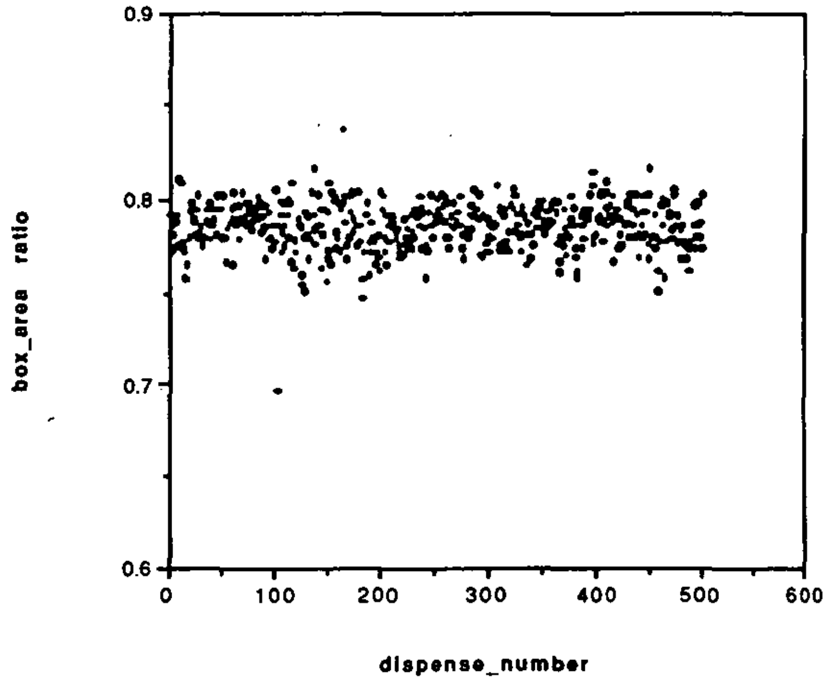
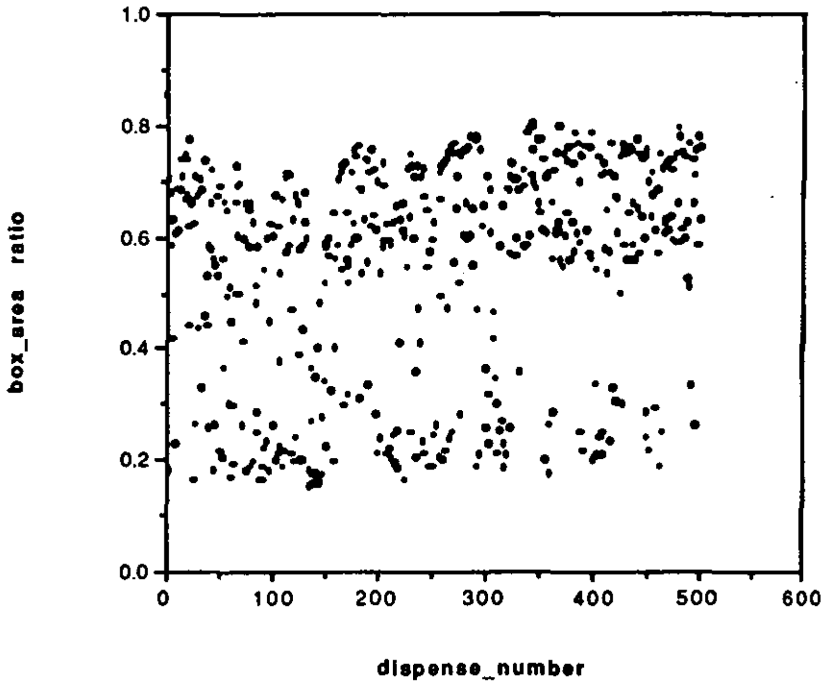


Fig 9.6 The box area ratio threshold, a good indication of the quality of the dispense



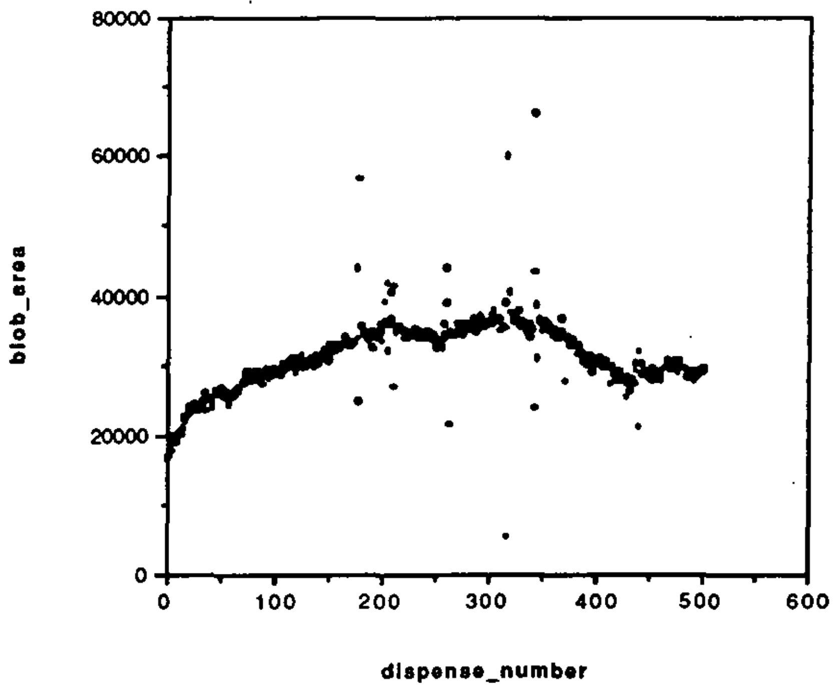
Graph 9.3 a. Good box area ratio performance for a given dispense experiment



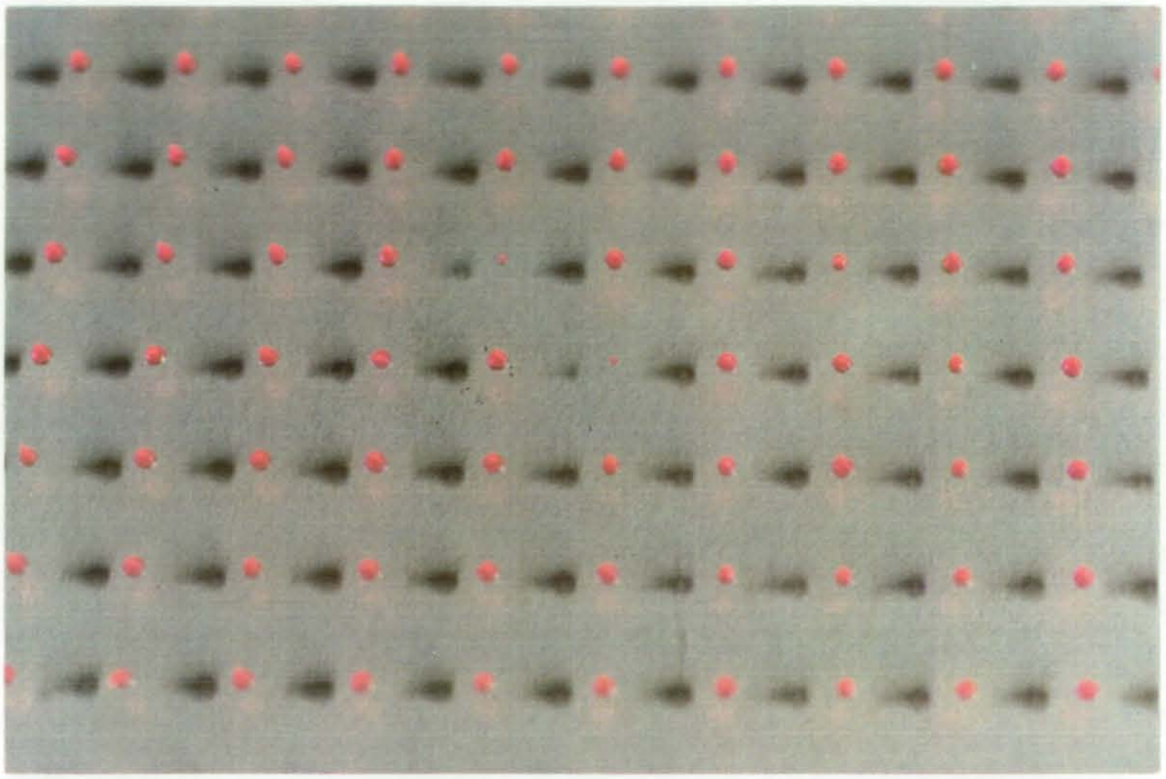
Graph 9.3 b. Bad box area ratio performance for a different dispense experiment

Voids in the adhesive occur from time to time and are characterised via a sharp increase in the blob area followed by a sharp decrease or total void of the area (see graph 9.4 and photograph 9.3). This is undesirable since the control of the blob size is important in ensuring that the surface components are securely fixed without using too much adhesive which would soil the board or the solder mountings. When the void occurs no adhesive is dispensed and the surface mount component will not be secure during the solder curing stage of manufacture. In the worst case the component will fail to be secured to the printed circuit board after soldering. Therefore the control process must recognise the onset of bubbles and take corrective measures. In this case it would take the syringe off board and clear the void before returning to normal execution.

When the airline that supplies the dispense unit is overloaded, the measured pulse height decays (graph 9.6). This condition must be monitored in order that the dispense process can be terminated and the fault rectified. This fault can result in erratic adhesive dispense performance (photograph 9.5).

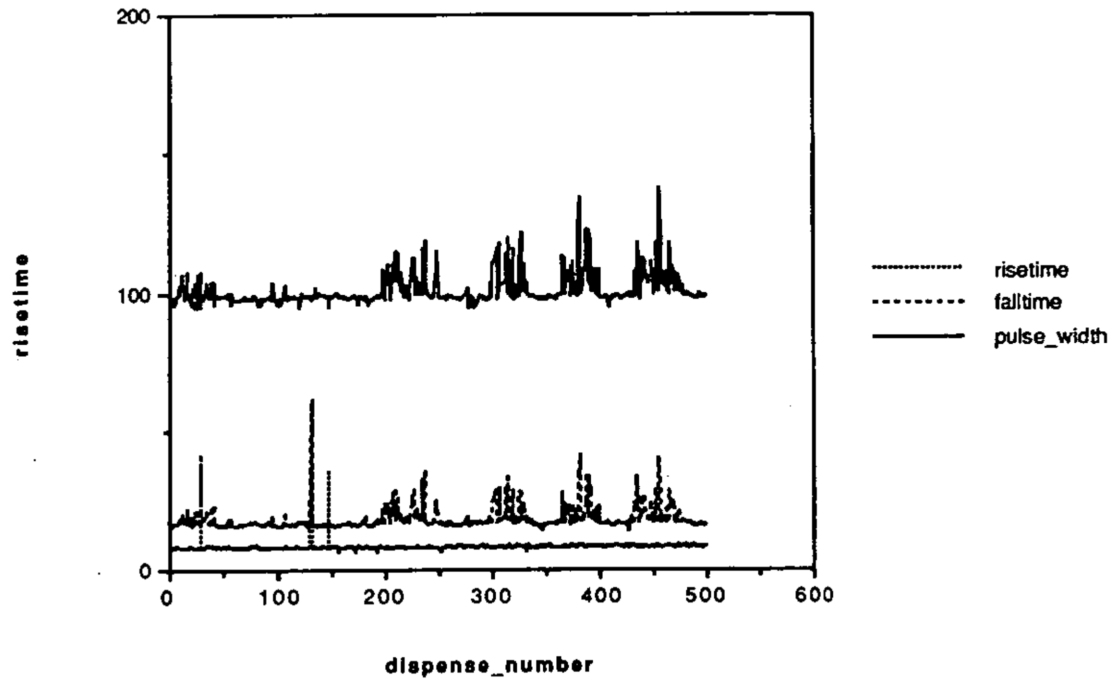


Graph 9.4 Open loop performance of a dispense experiment in which the properties of the bubbles can be seen

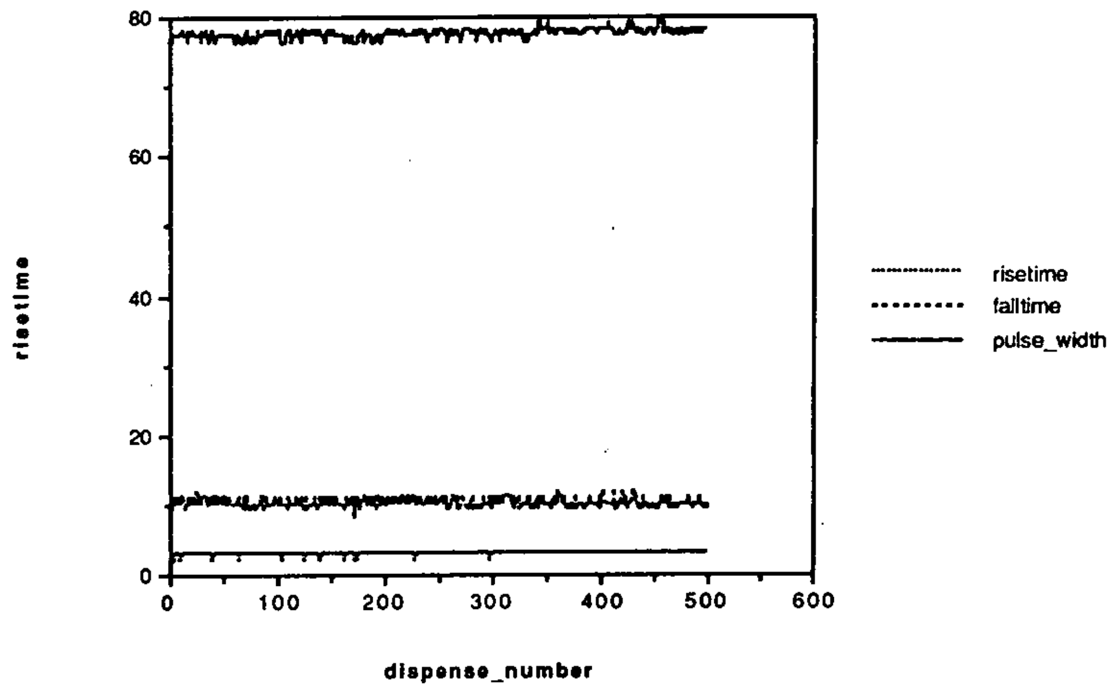


Photograph 9.3 Dispensed blobs which show the appearance of two poor dispenses following bubbles. Note, the random dispense order means that the increases (see photograph 9.1) prior to these dispenses are not adjacent.

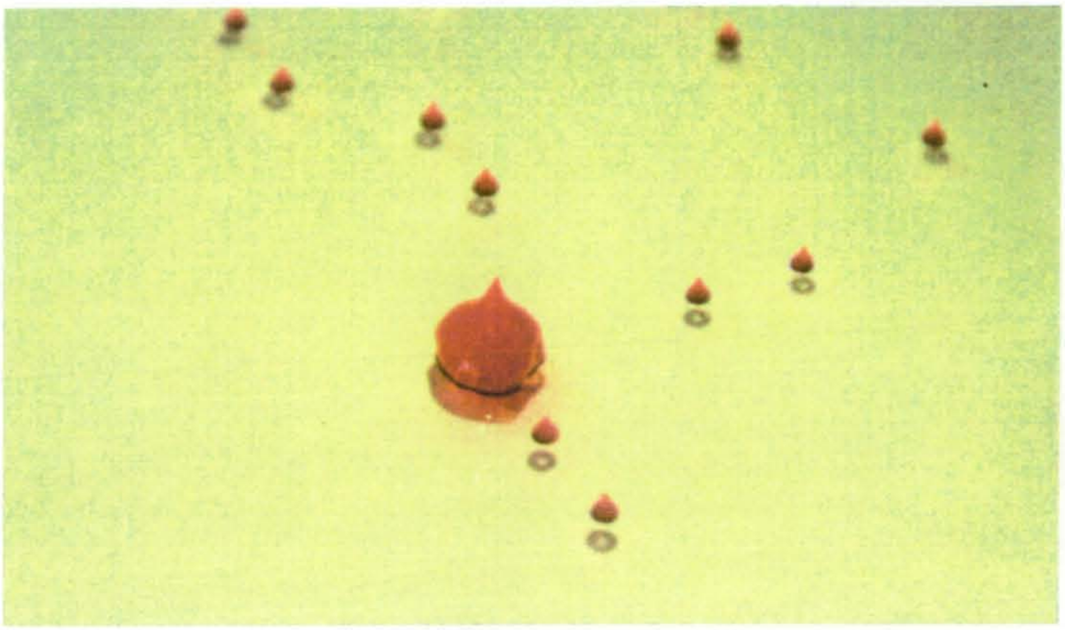
At times the solenoid valve of the pressure pulse regulator sticks in the open position, giving a pressure pulse with a larger fall time. The increased fall time allows a greater volume of adhesive to be dispensed. Graph 9.5a shows process data associated with a sticky solenoid valve. The graph shows the dispenses where the fall time increases by about 50 % of its normal value. If this problem is not corrected off line immediately the whole board will be fouled with excess adhesive. Photograph 9.4 shows the excess adhesive that may be dispensed when the solenoid valve starts to stick. Recognising this situation depends upon monitoring the fall time of the pressure pulse.



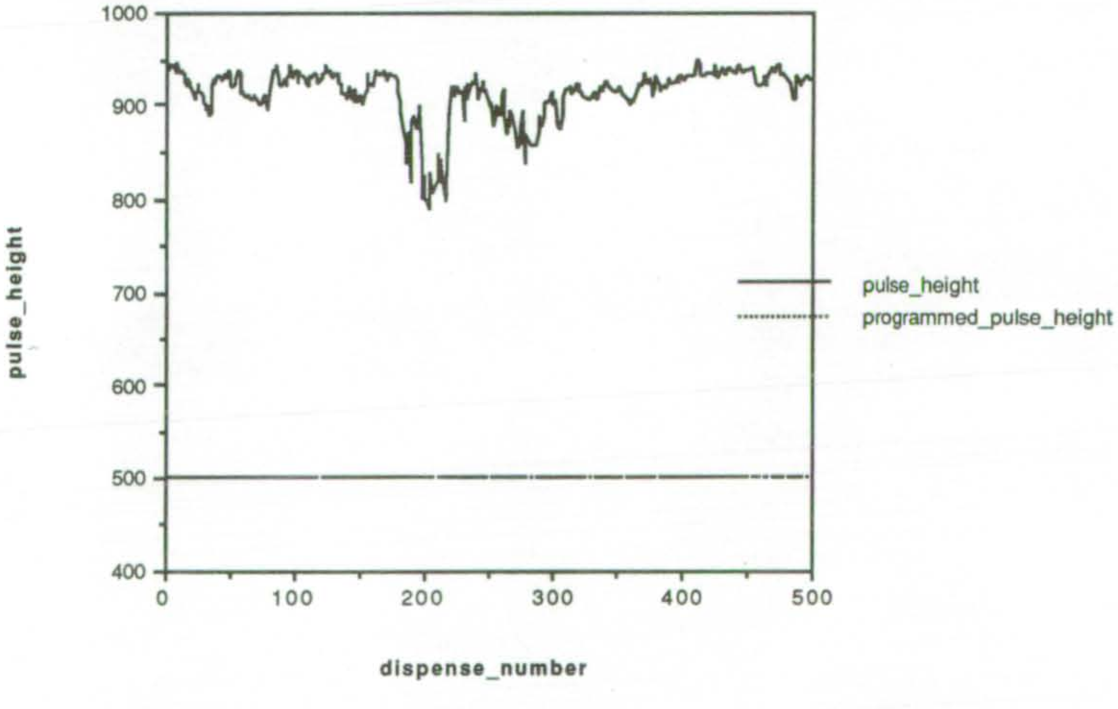
Graph 9.5 a. Rise time, fall time and pulse width performance of a dispense experiment that had a sticky solenoid valve problem



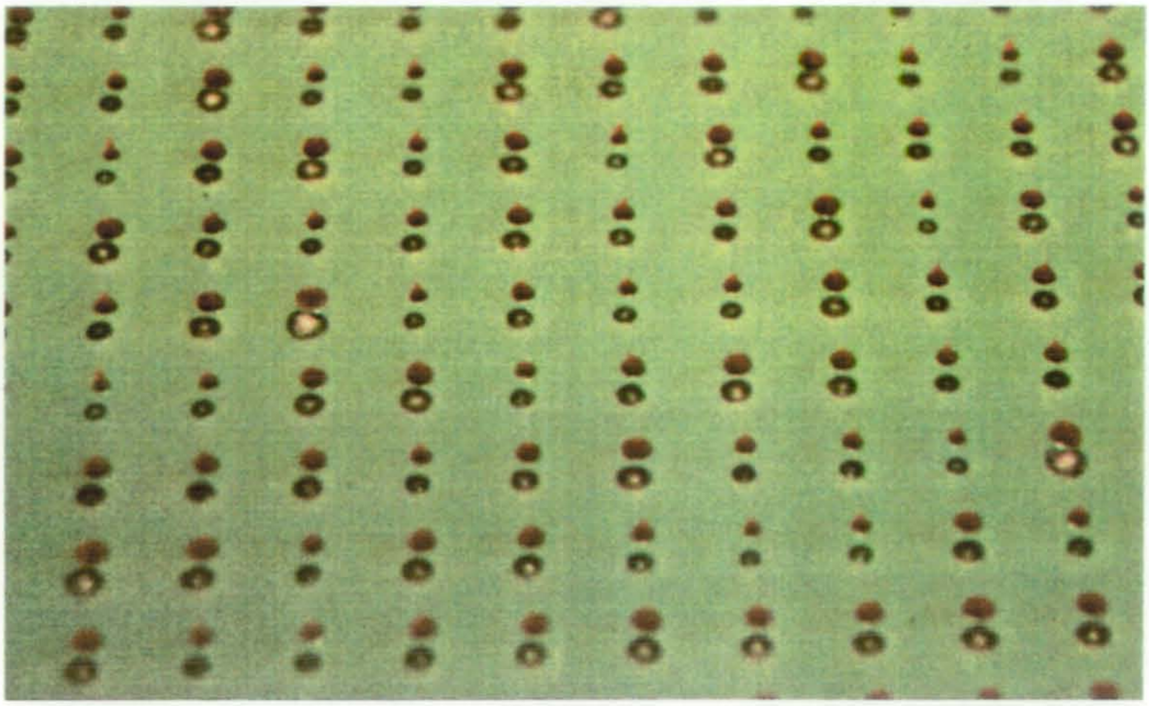
Graph 9.5 b. Rise time, fall time and pulse width performance of a normal dispense experiment



Photograph 9.4 Dispensed blobs when the solenoid valve was sticking. Note the excess adhesive that was dispensed when the solenoid valve failed catastrophically



Graph 9.6 The pulse height variation when extra load was applied to the air line

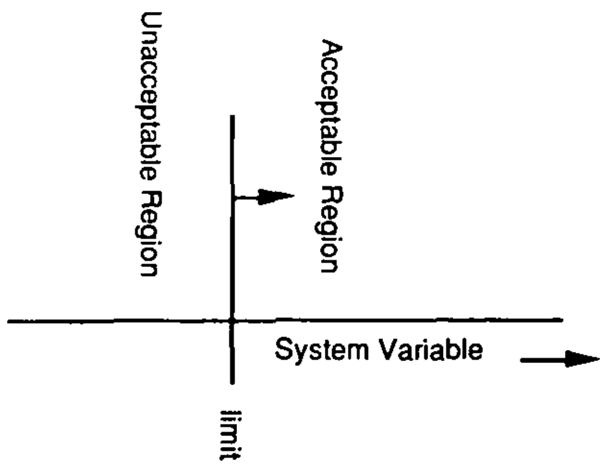


Photograph 9.5 Dispensed blobs when the extra load was applied to the air line. Note the material variation also contributes to the erratic blob sizes

Neural network implementations of control processes

Neural networks can model thresholds, bands and trends. These can be combined to construct reliable neural network implementations of controllers. A design of a neural network controller for the adhesive dispensing system is described below.

The feedback variables can be passed through a simple neural network controller that provides qualitative corrective actions. (The details of the neural network controller for the adhesive dispensing system are given in appendix G). The simplest case is represented by the box area ratio decision unit, which requires a single threshold device. Fig 9.7a shows the system variable (in this case the box area ratio thresholded in to the two regions). Fig 9.7b shows the single node that can implement this function.



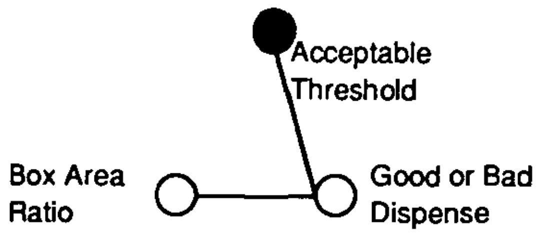


Fig 9.7 a. Thresholding of a single system variable, b.Neural network representation of the box area ratio threshold unit

The box area ratio is acceptable (output is +1) if it is larger than the given limit (0.78 in this case), while unacceptable (output is -1) if it is lower than the given limit. Similarly the pulse height drift can be monitored by a threshold unit that will flag the fault associated with the air line.

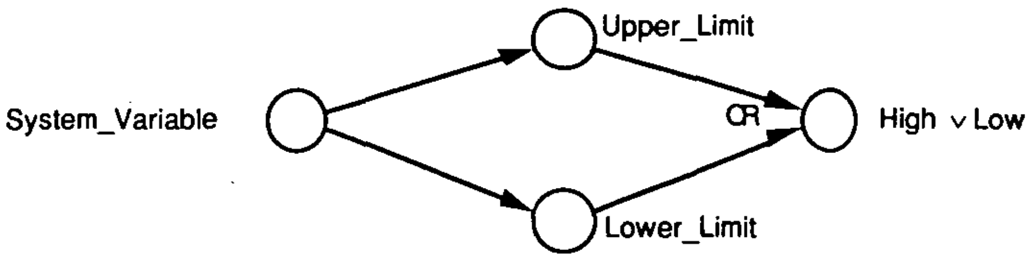


Fig 9.8 a. Neural network representation of the operator that keeps a single variable within given limits

The bang-bang controller of the blob area can be implemented in a neural network but requires two hidden nodes (fig 9.8a). The blob area has an acceptable value to within a given tolerance. This means that a region (or a band) of the system variable is acceptable surrounded by two regions that are unacceptable (see fig 9.8b). The output blob area decision unit is a logical or of the two hidden threshold units that provide the regional boundaries. No action (output is -1) is taken within the acceptable region of the system variable (blob area), while when the process strays outside of this region action (output is +1) is taken.

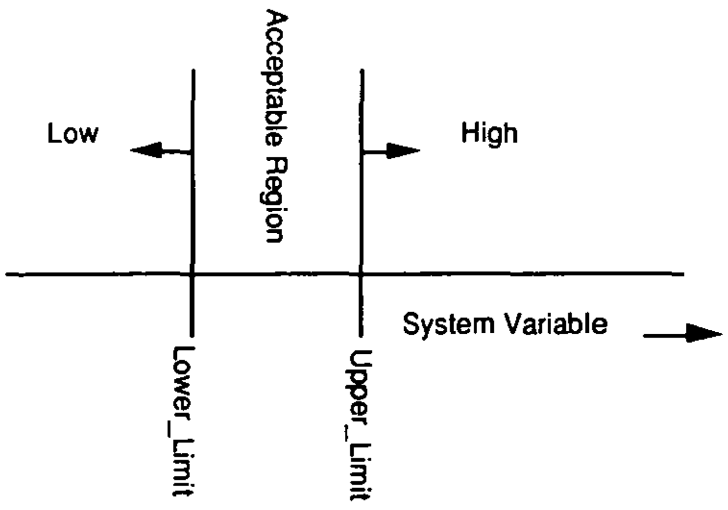


Fig 9.8 b. Segmentation of a single system variable into three regions

More complicated qualitative controllers require larger multilayered neural networks. Intelligent controllers are constructed via the addition of Boolean decision units within the hidden layers that provide the relevant control actions. Quantitative controllers are constructed via the addition of a decoding layer of weights from the qualitative decisions.

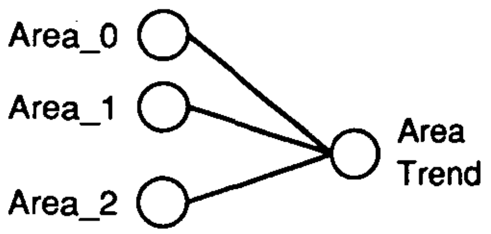


Fig 9.9 Trend analysis using a neural network unit

A neural network can be used to monitor blob area trend to anticipate an appearance of a bubble in the adhesive. The simplest approach would be to monitor the direction of change of the blob area of the latest dispenses, say the last three (see fig 9.9). A Boolean function of the inputs (in this case logical **and**) would ensure the correct qualitative response, take action or not, is made. A similar trend analysis of the fall time will recognise the sticky solenoid valve problem.

Training the neural network controller

Training a neural network requires data. Data can be obtained from the simulation models or from real process runs. The inputs to the network are the various process variables, while the outputs are the control actions that must be applied. The network is then trained automatically over all the data until satisfactory convergence is obtained.

The training sequence of the neural network controller requires that the network structure is well known and adequate for representing the transformation in question. The work presented in this thesis allows us to proceed and construct a neural network structure that is suitable for modelling the control problem.

The outputs of the neural network are Boolean values that provide decisions of say increase pressure or decrease pressure without specifying by how much or the actual real value.

A regional segmentation model of the blob area coupled with a box area ratio measure will provide a network structure suitable for most of the problems associated with this process, namely steady state process control and the detection of dragging of adhesive as well as the blockage of the syringe needle. Adding a trend analyser for the fall time and the blob area will take into account the remaining factors, namely the appearance of bubbles and the sticky solenoid valve problem. Fig 9.10 shows a suitable network structure for the control process. If enough is known explicitly about the process the weight values can be hand crafted. Otherwise the network must be trained automatically with the process data available. That is the network structure is tuned to the specific application.

Appendix G describes the various different neural network structures that were designed and the different methods used to train them. Including knowledge available about the control process rapidly improves the automated training algorithm's convergence performance.

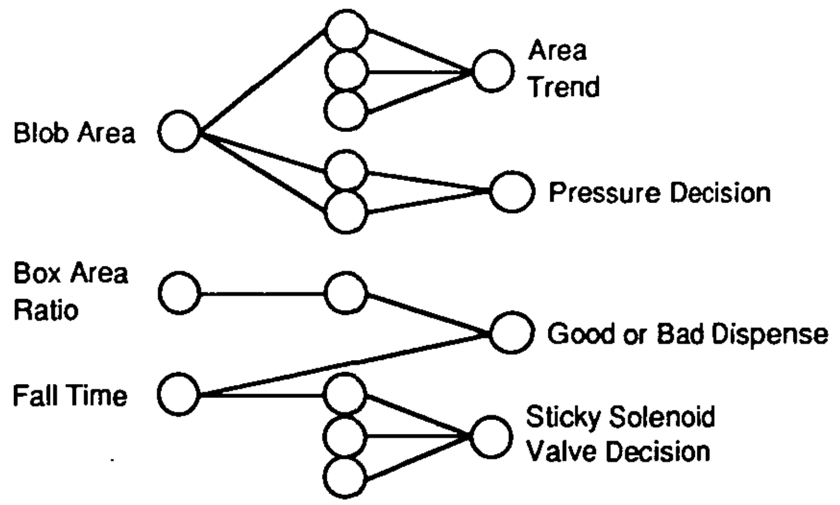


Fig 9.10 Neural model of an adhesive dispensing controller

The application of intelligent control actions such as the recovery from potentially catastrophic situations requires qualitative outputs from the controller. The signals from the controller essentially only initiate the required action, which is then carried out by specialised systems.

Intelligent control using neural networks

The Boolean layers of the neural network allow intelligent control actions to be implemented. This is illustrated by a simple two dimensional example. A trajectory must be maintained within a given tolerance (see fig 9.11). Corrective action is applied in the region outside the acceptable band. For intelligent control we require different corrective actions to be taken in different regions of the state space. Fig 9.11 shows a case where two separate actions must be taken in the regions A and B. The neural model of this example is shown in fig 9.12.

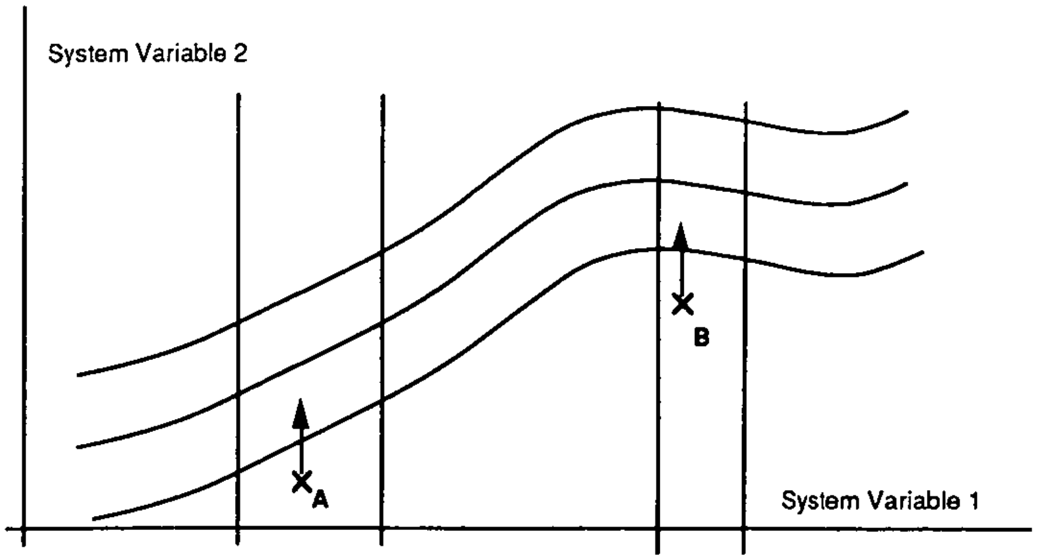


Fig 9.11 Simplified two variable example of intelligent control. Different control actions are required in different regions of the state space

The segmentation of the system variable requires four nodes while the trajectory tolerance boundaries require two nodes. The hidden nodes are required to model the Boolean transformation that provides the relevant control action.

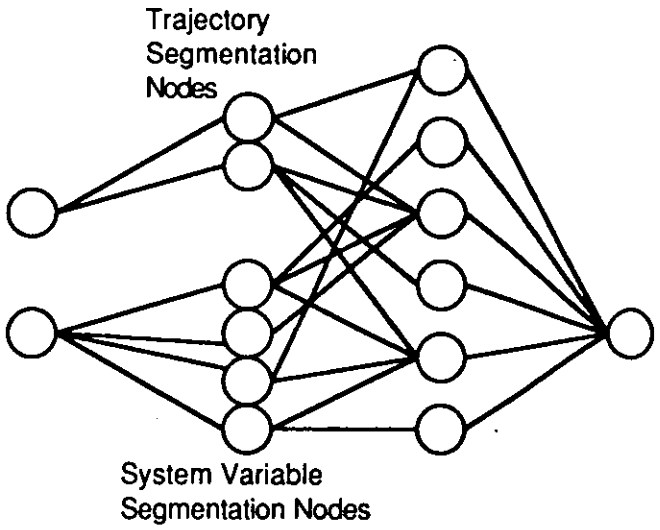


Fig 9.12 Neural model of intelligent controller shown in fig 9.11

Similarly intelligent control actions can be applied in the adhesive dispensing process. The bimodal warning operator possesses some of these properties. A warning signal is required when the process strays a given percentage from the ideal, so that preparations can be made for when the corrective actions are applied when the process strays outside the action error boundary. No warning signal is required when action is taking place. This is illustrated in fig 9.13. The neural model for this case is shown in fig 9.14. The four threshold nodes in the hidden layer represent the four warning and action error boundaries.

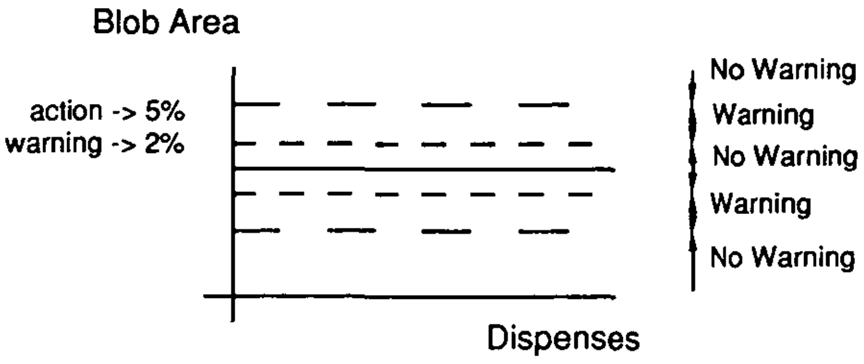


Fig 9.13 The bimodal warning operator

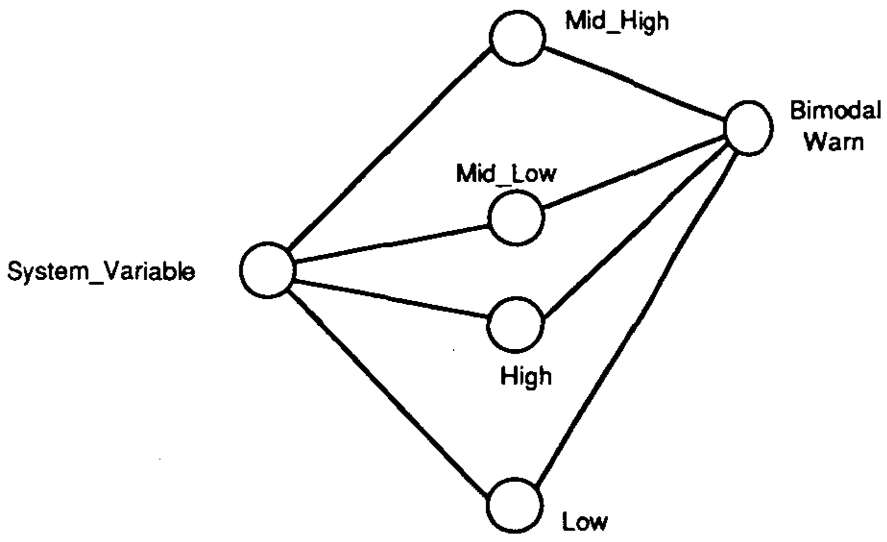


Fig 9.14 Neural implementation of the bimodal warning operator

Real output values

For more sophisticated controllers the control actions of the pressure pulse, namely increase or decrease pulse height is not a true qualitative action. A real valued quantity must be provided and so a decoding layer is necessary. At this point a design decision must be made as to how to achieve this output.

A fully quantised approach as discussed in chapter eight may be adopted, but this relies on the existence of an injective function from the input data to the output data, which may not exist with most control systems if we consider the effect of external variations on the system. The second approach is to maintain the qualitative decision to increase or decrease pressure but to decode it via a real valued weighted decoding unit which can be trained by the system (see fig 9.15 for a simplified network model). The output decoding unit is tuned to the data that is modelled, and so, if the data is representative of the system the best possible quantised decoding value will be found.

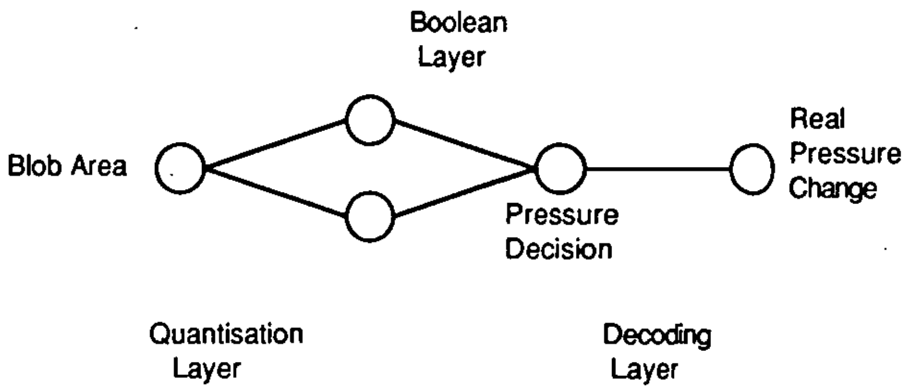


Fig 9.15 Simplified controller that provides real valued output signals

This fixed quantised decoding system is a rough model of the output parameters. An improved model can be sought by enhancing the system with a variable decoding unit. This allows the amount by which the pressure is varied to be increased or decreased. In this system there is an extra node which determines the magnitude of the decoding weight. The decoding weight is increased or decreased depending on the value of the change_decision node (fig 9.16). This development will be pursued further in the future. The structure of the neural network for this case is shown in fig 9.16.

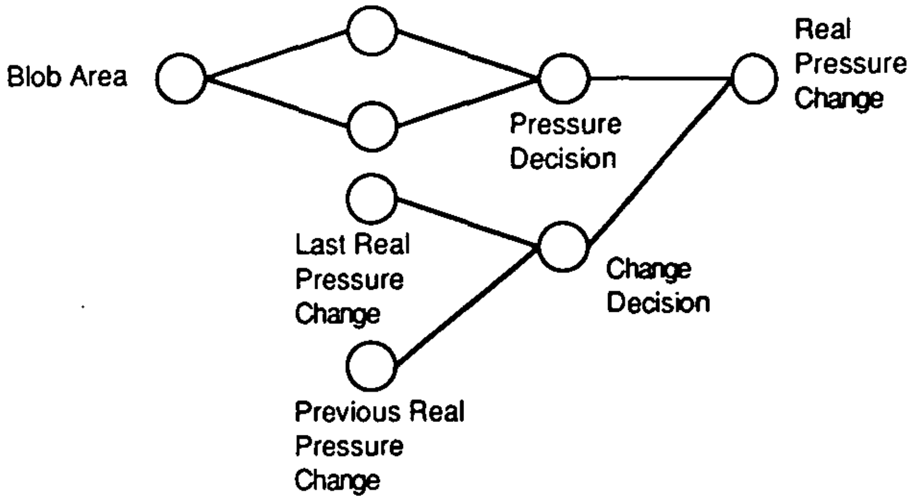


Fig 9.16 Network architecture for a controller that gives a variable pressure change

Summary

This chapter has discussed the design and implementation of a neural network controller for an adhesive dispensing system. The properties of the system were discussed and the various process characteristics and process faults highlighted.

It was shown that neural network techniques could implement the thresholding and banding of the process variables which allowed the relevant control signals to be output and the relevant process faults to be flagged. Ideas for implementing intelligent control actions using neural networks were discussed.

The neural network that was designed solved the adhesive dispensing control problem. The designed neural network controller was predictable and reliable since its behaviour was well defined over the whole input space. Appendix G shows the performance of the neural network controller.

Chapter 10. Conclusions and Direction of Future Work

Summary of Thesis

This thesis has addressed the question of reliable neural network design and a study of feed forward neural networks has been given. The training of neural networks was discussed. Particularly the backpropagation algorithm has been examined and its use in training real as well as Boolean transformation networks has been studied.

The treatment of the hidden nodes as Boolean transformations has moved neural network techniques into the area of reliable systems. The introduction of parallel nodes enabled the hidden layers to be structured and the performance of the networks to be guaranteed. The parallel nodes were proposed as atomic elements in a general knowledge based representation of neural systems. These systems were extensively studied and compared to standard feedforward systems.

As well as the computational advantage of the parallel node system (the reduction of the number of weights which lead to the reduced load on the learning algorithm) the significant advantage of the new proposals was the reliability of the neural network behaviour and the ability to interpret the neural network structure in an atomic manner.

On the basis of the investigations into neural network structure and behaviour several results on network size and topology were presented. These results are crucial in the design of neural network systems. Only after a suitable size and topology of a network has been chosen do the automatic training algorithms provide the weight values of the network that model the input/ output data.

Given the knowledge about the constraints on neural network size and topology a general design methodology was proposed. The methodology provides a prescriptive scheme of action for designing neural models of input/ output systems, whether they be real or Boolean values. Finally the work was applied to the design of a real time neural network controller of an adhesive dispensing system. The application demonstrated the importance of the design of reliable systems whose behaviour is fully understood. This allowed important system characteristics that were already known to be included in the network structure so reducing the training phase of the neural network.

Future work

Future work will be in the following areas;

i. Training algorithms,

ii. Knowledge representations,

iii. Design tools,

iv. Real applications.

Training algorithms

The backpropagation algorithm has pushed neural networks to the forefront of public attention. As well as the advantage of parallel representation and distributed activity over simple units, it is the automated training algorithms that make neural network representations attractive. The experiments described in part II and appendices A, B, C and F of this thesis show that training is still a long and difficult process. When the training performance is so dependent on the possibly random start point, it can be seen that more study is required. The application of genetic algorithms is likely to be one of the most fruitful paths. The genetic algorithm will be able to propagate multiple start points which can then be optimised by a backpropagation technique. The parallel nature of the genetic algorithm will allow this technique to be fully exploited as multiprocessor systems become more powerful and sophisticated.

Knowledge representations

The key to understanding the behaviour of neural network models is to have a well defined structure and interaction of nodes in the network. The parallel nodes represent a starting point. The nodes are well defined, that is parallel and there is no interaction between pairs of parallel nodes. Hyperpolygonal systems as well as systems that allow more complex interaction between the nodes will be developed in the future. Encapsulating the function and behaviour of networks and subnetworks in much the same manner as object orientated systems will allow the hierarchical design of neural network systems without losing their inherent parallelism.

The representational capabilities of neural systems can also be extended by the use of Fuzzy and probabilistic modelling techniques. The development of fuzzy and stochastic neural models are discussed by Sprecht '90. The use of knowledge representations in these systems would formalise system performance and aid understanding of the behaviour of the neural networks.

Design tools

As more complex systems are modelled by neural network structures, automated design tools must be made available. Automated interpreters for converting from neural network to Boolean representations and vice versa would be a start point. Automated network encapsulation systems would allow rapid prototyping and development of designed neural systems. The reliability of the systems would be maintained by the use of parallel nodes as well as other specified knowledge structures.

Applications

As neural network techniques develop their applications to real world systems will increase. Significant engineering applications are possible as the question of neural network reliability has been addressed by this thesis. The control problem discussed in chapter nine is a case in point. The low level nature of real time signal processing makes the neural network systems which have been implemented in hardware an ideal solution.

The development of the structured knowledge representation of neural systems will allow higher level applications of neural networks. General neural network computational systems may then be built on this structured approach to distributed neural computation.

References

Abu Mustafa, Y.S. & St Jacques, J.M.(1985), Information Capacity of the Hopfield model, IEEE Transactions on Information Theory, vol 31, pp 461- 464.

Aleksander, I. & Morton, H.(1990), An Introduction to Neural Computing, Chapman and Hall, London.

Amari, S.I.(1967), A Theory of Adaptive Pattern Classifiers, IEEE Transactions on Electronic Computers, vol EC 16, pp 299- 307.

Amari, S.I.(1977), A Mathematical Approach to Neural Systems, Arbib, M.A. & Metzler, J.(Eds): Systeme Neuroscience, pp 67- 117, Academic Press, New York.

Amari, S.I. & Arbib, M.A.(1977), Competition and Cooperation in Neural Nets, Arbib, M.A. & Metzler, J.(Eds): Systeme Neuroscience, pp 119- 165, Academic Press, New York.

Antsaklis, P.J.(1992), Neural Networks in Control Systems, IEEE Control Systems Magazine, vol 12, pp 8- 10.

Arai, M.(1989), Mapping Abilities of Three Layer Neural Networks, International Joint Conference on Neural Networks, pp 419- 423.

Aynsley, M., Peel, D. & Morris, A.J.(1989), A Real Time Knowledge Based System for Fermentation Control, American Control Conference, pp 2239- 2244.

Barbiarz, A.J.(1989), Adhesive Dispensing for Surface Mount Assembly, Printed Circuit

Assembly, July, pp 8- 11.

Barto, A.G., Sutton, R.S. & Anderson, C.W.(1983), Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems, IEEE Transactions on Systems Man and Cybernetics, vol 13, pp 834- 846.

Barto, A.G.(1990), Connectionist Learning for Control, Neural Networks for Control, Sutton R.S. & Werbos, P.J.(editors), MIT Press, Cambridge MA.

Baum, E.B.(1988), On the Capabilities of Multilayer Perceptrons, Journal of Complexity, vol 4, pp 193- 215.

Birdsall, D. & Cipolla, C.M.(1979), The Technology of Man, Wildwood House Ltd, London.

Birkhoff, G. & Bartee, T.(1970), Modern Applied Algebra, McGraw-Hill Book Company, London.

Birkhoff, G. & Maclane S.(1977), A Survey of Modern Algebra, Macmillan.

Block, H.D.(1970), A review of "Perceptrons: an introduction to computational geometry", Information and Control, 17, pp 501-522.

Boole, G.(1847), The Mathematical Analysis of Logic, Macmillan, Cambridge.

Boole, G.(1958), An Investigation of the Laws of Thought, Dover Publications, New York.

Bourne, D.A.(1986), CML- A Meta Interpreter for Manufacturing, AI Magazine, vol 7, pp 86- 96.

Bourne, D.A.(1987), The Automated Craftsman- Preliminary Research, CMU-RI-TR-87-22.

Boyer, C.B. & Merzbach, U.C.(1989), A History of Mathematics, Second Edition, John Wiley and Sons, Chichester.

Bridgeman, K.(1989), Dispensing Liquid Adhesives, Engineering, June.

Bronowski, J.(1973), The Ascent of Man, British Broadcasting Corporation, London.

Budge, T., Craven, S., Duran , S., Pearson, J.T., Welch, R. & Wossun, M.(1990), PARSEC-Process Analysis with Recipe Support for Etcher Control, IEEE Transactions on Semi-conductor Manufacturing, vol 3, pp 28- 32.

Carroll, L., Bartley,W.W.(editor),(1977), Symbolic Logic, The Harvester Press Limited, Hassocks , England.

Carroll, L.(1958), Symbolic Logic and the Game of Logic, Dover Publications, New York.

Caudhill, M.(1988), The Polynomial Adaline Algorithm, Comput Language, vol Dec, pp 53-59.

Chandraker, R., West, A. A. and Williams, D. J.(1989), Knowledge based Control of Adhesive dispensing for Surface Mount Device Assembly, IEEE Components Hybrids and Manufacturing Technology Symposium, September.

Chandraker, R., West, A. A. and Williams, D. J.(1990), Intelligent control of Adhesive Dispensing, IJCIM, Special issue in Intelligent Control, vol. 3, No 1, pp. 24-34.

Clark, R.W.(1985), Works of Man, Century Publishing, London.

Clocksin, W. & Mellish, C.(1987),Programming in Prolog, 3rd Ed., Springer-Verlag.

Clarke, A.A.(1986), A Three Level Human Computer Interface Model, International Journal of Man Machine Studies, Vol 24, pp 503- 517.

DeMorgan, A.(1847), Formal Logic, Taylor and Walton, London.

Dolan, C.P. & Dyer, M.G.(1987), Towards the Evolution of Symbols, Proceedings of the Second International Conference on Genetic Algorithms, pp 123- 131.

Efstathiou, J.(1985), Rule Based Process Control, Expert Systems in Process Control and Optimisation, Proceedings of Seminar Dec 1985, Unicom Seminars.

Efstathiou, J.(1989), Rule Based Process Control, Longmans, London.

Feldman, J.A.(1982), Dynamic Connections in Neural Networks, Biological Cybernetics, vol 46, pp 27- 39.

Feldman, J.A. & Ballard, D.H.(1982), Connectionist Models and their Properties, Cognitive Science, 6, pp 205- 254.

Feldman, J.A., Fanty, M.A., Goddard, N.H. & Kenton, J.L.(1988), Computing with Structured Connectionist Networks, Communications of the ACM, 31, pp 170- 187.

Forbes, R.J. & Dijksterius, E.J.(1963), A History of Science and Technology, Vol. 1 & 2, Penguin Book Ltd, Harmondsworth.

Gallant, S.I.(1988), Connectionist Expert Systems, Communications of the ACM, 31, pp 152- 169.

Geszti, T.(1990), Physical Models of Neural Networks, World Scientific.

Grant, E. & Zhang, B.(1989), A Neural Net Approach to Supervised Learning of Pole Balancing, Proc. 4th International Symposium on Intelligent Control, 25-27 September, Albany, New York, pp 123- 129.

Hertz, J., Krogh, A. & Palmer, R.G.(1991), Introduction to the Theory of Neural Computation, Addison Wesley Publishing Company, Wokingham, United Kingdom.

Hinde, C.J.(1974), Heuristic techniques applied to an industrial situation, Ph.D. thesis Brunel University.

Hinde, C.J.(1990), A Comparative Review of Neural Nets and Rule Induction Systems, Department of Computer Studies Technical Report, LUT TR 575.

Hinton, G.E.(1986), Learning Distributed Representations of Concepts, Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Amherst, pp 1- 12.

Hinton, G.E.(1989), Connectionist learning procedures, Artificial Intelligence, 40, pp 185- 234.

Hinton, G.E.(1990), Mapping Part Whole Hierarchies into Connectionist Networks, Artificial Intelligence, vol 46, pp 47- 76.

Hopfield, J.J. & Tank, D.W.(1986), Computing with Neural Circuits: A Model, Science, 233, pp 625-633.

Hopfield, J.J.(1984), Neurons with Graded Response have Collective Computational Properties like those of Two-state Neurons, Proceedings of the National Academy of Science USA, vol 81, pp 3088- 3092.

Huang, S.C. & Huang, Y.F.(1991), Bounds on the Number of Hidden Neurons in Multilayer Perceptrons, IEEE Transactions on Neural Networks, 2, pp 47- 55.

Hwang, K. & Briggs, F.A.(1985), Computer Architecture and Parallel Processing, McGraw Hill Book Company, London.

Jacyna, G.M. & Malaret, E.R.(1989), Classification Performance of a Hopfield Neural Network Based on a Hebbian-like Learning Rule, IEEE Transactions on Information Theory, vol 35, pp 263- 280.

Karsai, (1992), Journal of Intelligent Manufacturing: Special Issue on Neural Networks, pp 56- 65.

Keynes, J.N.(1906), Studies and Exercises in Formal Logic, Macmillan, London.

Keynes, J.M.(1921), A Treatise on Probability, Macmillan, London.

Kolb, B. & Whishaw, I.Q.(1980), Fundamentals of Human Neuropsychology, W.H.Freeman and Company.

Kowalski, R.(1980), Logic for Problem solving, North-Holland.

Knapp,L.(1987), SMD adhesives and Dispense systems, Industrial Report.

Kraft, L.G. & Campagna, D.P.(1990), A Summary Comparison of CMAC Neural Network and Traditional Adaptive Control Systems, in Neural Networks for Control, (editors Miller,W.T., Sutton, R.S. & Werbos, P.J.), The MIT press, London.

Kung, S.Y. & Hwang, J.N.(1988), An Algebraic Projection Analysis for Optimal Hidden Units Size and Learning Rates in Back Propagation Learning, IEEE International Conference on Neural Networks, pp 363- 370.

Lilley, S.(1965), Man Machines and History, Lawrence and Wishart, London.

Lipmann, R.P.(1987), An Introduction to Computing with Neural Nets, IEEE ASSP Mag, 4, pp 4- 22.

Massey, J.G., Wickman, J.L. & Cook, D.F.(1990), Incorporating Statistical Process Control into a Lumber Manufacturing System, AI in Engineering, pp 155- 166, Boston.

Materna, T.(1987), "Neural networks enter high speed marketplace", Computer Technology Review, vol. 7, No 7, June.

McClelland, J.L. & Rumelhart, D.E.(1985), Distributed Memory and the Representation of General and Specific Information, Journal of Experimental Psychology: General, vol 114, pp 159- 188.

McCulloch & Pitts,(1943), A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, vol 5, pp 115- 133.

Messom, C.H.(1992), Dependencies in the hidden layer of neural systems, Dept. Computer Studies, Loughborough University of Technology Internal Report TR 674.

Messom, C.H., Hinde, C.J., West, A.A. & Williams, D.J.(1992), Designing Neural Networks for Manufacturing Process Control Systems, IEEE International Symposium on Intelligent Control, Glasgow.

Mikami, Y.(1974), The Development of Mathematics in China and Japan, Chelsea Publishing Company, New York.

Miller, W.T., Hewes, R.P., Glanz, F.H. & Kraft, L.G.(1990), Real-time Dynamic Control of an Industrial Manipulator Using a Neural Network Based Learning Controller, IEEE Transactions on Robotics and Control, vol 6, pp 1- 9.

Minsky, M.L. and Papert, S.(1969), Perceptrons, M.I.T. press.

Mirchandani, G. & Cao, W.(1989), On Hidden Nodes for Neural Nets, IEEE Transactions on Circuits and Systems, 36, pp 661- 664.

Morgan, A.J.(1990), Real Time Expert Systems in the Cogsys Environment, in Research and Development in Expert Systems VII, Proceedings of Expert Systems 90, London, September 1990, pp 104- 115.

Opdahl, P.O.(1989), EPAK- Expert System for Paper Quality, Gensyn Users Society, Fall Meeting, Cambridge, MA, October 1989.

Pei-Yan, Z., & Bin, L.(1990), A Fuzzy Control Method for a Cylindrical Grinding Process, CAPE6 Conference, London, November 1990, pp 491- 493.

Pollack, J.B.(1990), Recursive Distributed Representations, Artificial Intelligence, vol 46, pp 77- 106.

Rosenblatt, F.(1962), Principles of neurodynamics, Spartan books.

Rumelhart, D.E., McClelland, J.L. & PDP Research Group.(1986), Parallel Distributed Processing;- Explorations in the Microstructure of Cognition, 1 & 2, MIT Press.

Russell, B.(1919), Introduction to Mathematical Philosophy, Allen and Unwin, London.

Russell, B.(1903), The Principals of Mathematics, Allen and Unwin, London.

Sakurai, M., Wakimoto, K., Nakajima, R. Maki, A. & Sakai, A.(1989), Operation Control of a Blast Furnace by Artificial Intelligence, Iron and Steel Manufacturing, November, pp 59-67.

Sejnowski, T.J. & Rosenberg, C.R.(1987), Parallel Networks that Learn to Pronounce English Text, Complex Systems, vol 1, pp 145-168.

Shapiro, E.Y. & Stirling, L.(1985), The Art of Prolog: Advance Programming Techniques, M.I.T. Press.

Sloman, M. & Kramer, J.(1987), Distributed Systems and Computer Networks, Prentice Hall International, London.

Sprecht, D.F.(1967), Generation of Polynomial Discriminant Functions for Pattern Recognition, IEEE Trans Electron Comput, vol EC 16, pp 308- 319.

Sprecht, D.F.(1990), Probabilistic Neural Networks and the Polynomial Adaline as

Complementary Techniques for Classification, IEEE Transactions on Neural Networks, vol 1, pp 111- 121.

Stengel, R.F.(1991), Intelligent Failure-Tolerant Control, IEEE Control Systems Magazine, Vol 11, No4, pp 14- 23.

Tani, J., Hirobe, T., Niida, K., Koshijima, I. & Murakami, H.(1989), New Learning Algorithm for Rule Extraction by Neural Network and its Application, AAAI.

Tank, D.W. & Hopfield, J.J.(1986), Collective Computation in Neuronlike Circuits, pp 62-70.

Tank, D.W. & Hopfield, J.J.(1986), Computing with Neural Networks: A Model, Science, Vol 233, pp 625- 633.

Turban, E.(1988), Decision Support and Expert Systems, MacMillan Publishing Company, London.

Venn, J.(1894), Symbolic Logic, Macmillan, London.

Vogl, T.P., Mangis, J.K., Rigler, A.K., Zink, W.T. & Alkon, D.L.(1988), Accelerating the Convergence of Backpropagation, Biological Cybernetics, vol 59, pp 257- 263.

Wasserman, G.D.(1972), Molecular Control of Cell Differentiation and Morphogenesis: a systematic theory, Marcel Dekker, New York.

Wasserman, P.D. & Schwartz, T.(1987), Neural Networks:- Why is Everybody Interested in them Now; Part 1, IEEE Winter 1987, pp 10-15.

Wasserman, P.D. & Schwartz, T.(1988), Neural Networks:- Why is Everybody Interested in them Now; Part 2, IEEE Spring 1988, pp 10-15.

Warwick, K. & Tham, M.T., Editors,(1991), Failsafe Control Systems: Applications and Emergency Management, Chapman and Hall, London.

West, A.A., Chandraker, R., Williams, D.J., Mulvaney, D.J.(1988), Hybrid Representations of Real-time Control Rules for Manufacturing Process Control in Electronics Manufacture, Proceedings of the IEEE Symposium on Intelligent Control, Arlington, Virginia, September.

West, A.A.(1992), ACME Grant GR/ F 71973 & GR/ G 37101, Final Report.

Williams, D.J.(1990), The hidden Problems of Surface Mount Device Assembly, Assembly Automation, vol 9, No 2, p 59- 60.

Williams, D.J., West, A.A., Chandraker, R.(1989), Knowledge Based Control of Adhesive Dispensing for SMD Assembly, SERC ACME Grant GR/E 40040 Final Report.

Willshaw, D.J.(1981), Models of Distributed Asociative Memory, Doctoral Dissertation, University of Edinburgh.

Wright, P. K., Pavlakos, E. & Hansen, F.(1991), "Controlling the Physics of Machining on a New Open-Architecture Manufacturing System", ASME Winter Annual Meeting.

Zhang, B. & Grant, E.(1991), A Neural Net Approach to Adaptive State Space Partition for Learning Control.

Appendices

Appendix A1. Start Point Experiment

Aim:

To identify the effect of the start point on neural network training algorithm convergence. That is the effect of different initial neural network weight configurations on training algorithm performance.

Method:

Four data sets where selected to examine the performance of the backpropagation training algorithm over different start points. These where odd and even parity with three and four inputs.

Three input even parity	Three input odd parity
p(-1,-1,-1,1).	p(-1,-1,-1,-1).
p(-1,-1,1,-1).	p(-1,-1,1,1).
p(-1,1,-1,-1).	p(-1,1,-1,1).
p(-1,1,1,1).	p(-1,1,1,-1).
p(1,-1,-1,-1).	p(1,-1,-1,1).
p(1,-1,1,1).	p(1,-1,1,-1).
p(1,1,-1,1).	p(1,1,-1,-1).
p(1,1,1,-1).	p(1,1,1,1).

Four input even parity

p(-1,-1,-1,-1,1).

p(-1,-1,-1,1,-1).

p(-1,-1,1,-1,-1).

p(-1,-1,1,1,1).

p(-1,1,-1,-1,-1).

p(-1,1,-1,1,1).

p(-1,1,1,-1,1).

p(-1,1,1,1,-1).

p(1,-1,-1,-1,-1).

p(1,-1,-1,1,1).

p(1,-1,1,-1,1).

p(1,-1,1,1,-1).

p(1,1,-1,-1,1).

p(1,1,-1,1,-1).

p(1,1,1,-1,-1).

p(1,1,1,1,1).

Four input odd parity

p(-1,-1,-1,-1,-1).

p(-1,-1,-1,1,1).

p(-1,-1,1,-1,1).

p(-1,-1,1,1,-1).

p(-1,1,-1,-1,1).

p(-1,1,-1,1,-1).

p(-1,1,1,-1,-1).

p(-1,1,1,1,1).

p(1,-1,-1,-1,1).

p(1,-1,-1,1,-1).

p(1,-1,1,-1,-1).

p(1,-1,1,1,1).

p(1,1,-1,-1,-1).

p(1,1,-1,1,1).

p(1,1,1,-1,1).

p(1,1,1,1,-1).

The different start points that where used in this experiment where generated by a seeded pseudorandom scheme. A three input and four input fully connected structure with three hidden nodes and one output node network structure requires sixteen and twenty five weights respectively. The first weight is generated from the seed value via the following function;

Weight= (Seed - 505)/1000,

Nextseed= (Seed * 997 * 101) mod 1009,

while the next weight is generated from the next seed. This process continues iteratively for all the weights in the network.

The learning rate that was employed was 0.01, while the temperature value was 0.1. Seed values from 3 to 51 were used in this experiment. Each initial configuration was trained over the odd and even parity data separately for 2000 iterations. The results are presented in the graphs A1.1 a- d and table A1.1 below.

Several of the typical convergence paths are displayed in graphs A1.2 a- g.

Results:

As shown in the graphs A1.1 a- d the sum squared error convergence performance of the backpropagation algorithm is highly dependent on the start point, the initial configuration of the neural network. For the three input case it is seen that a third (17 for even parity and 18 for odd parity) of the neural networks converge within the 1000 iterations examined. For the four input case it is seen that a sixteenth (3 for even parity and 3 for odd parity) of the neural networks converge within the 1000 iterations while there is a large variation in sum squared error values for the other start points.

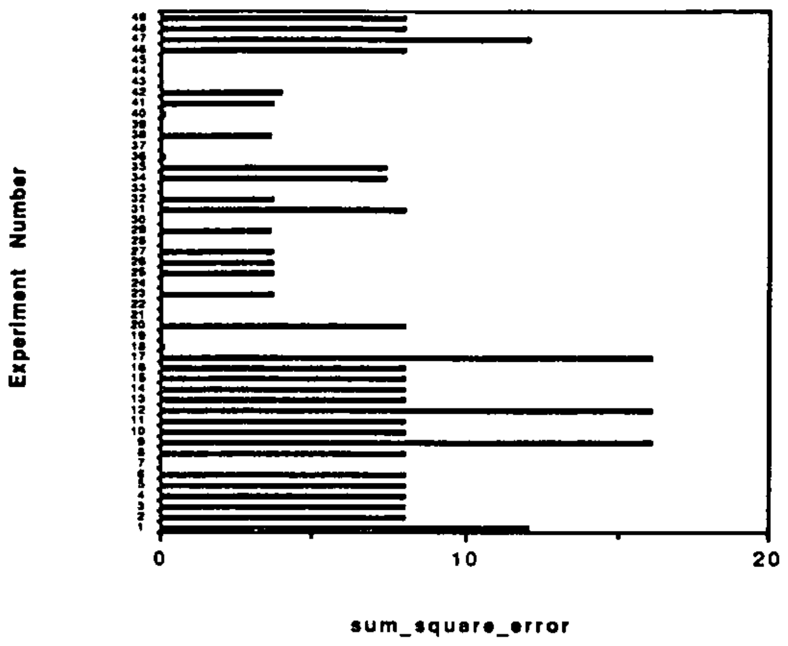
Experiment	Mean	Variance
3 input even parity	5.238	4.566
3 input odd parity	4.920	4.483
4 input even parity	12.405	8.691
4 input odd parity	13.290	10.181

Table A1.1 The mean and variance of the sum square error convergence performance of the four sets of experiments

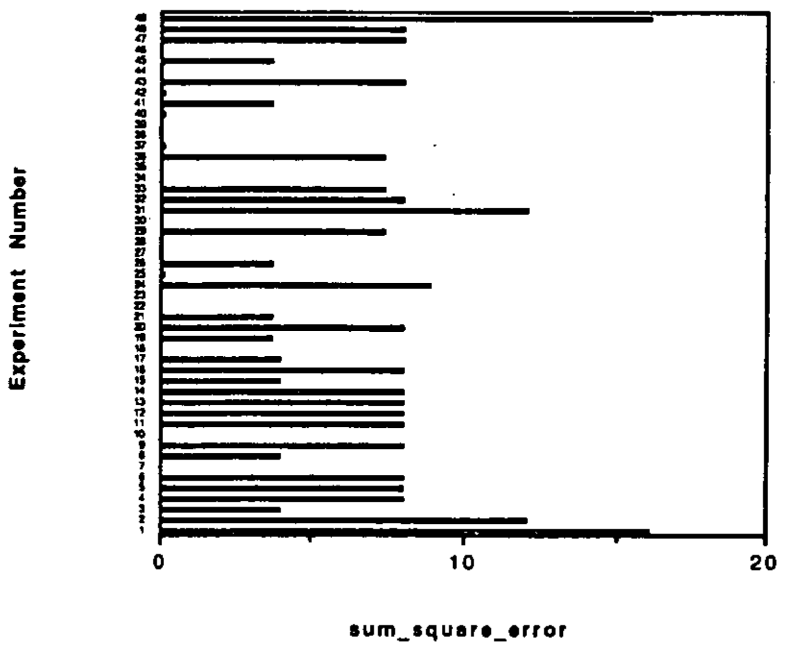
Analysis:

The typical sum squared error convergence paths that are shown in graphs A1.2 a- g vary greatly. This is dependent on the initial configuration of the neural networks and not the training data. The great variation in convergence cannot be explained by the four training data, but is a function of the initial configurations of the neural networks (the start points).

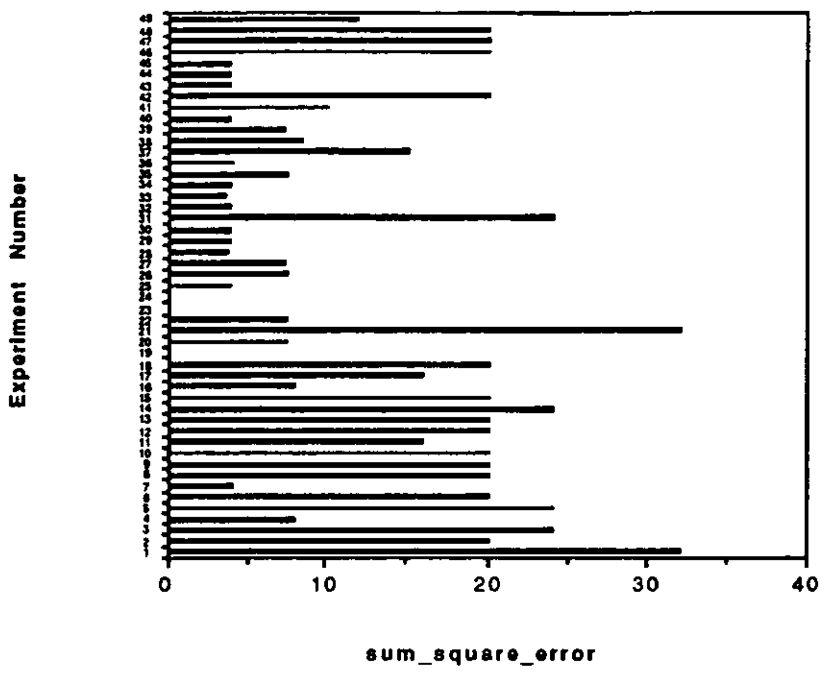
Graphs:



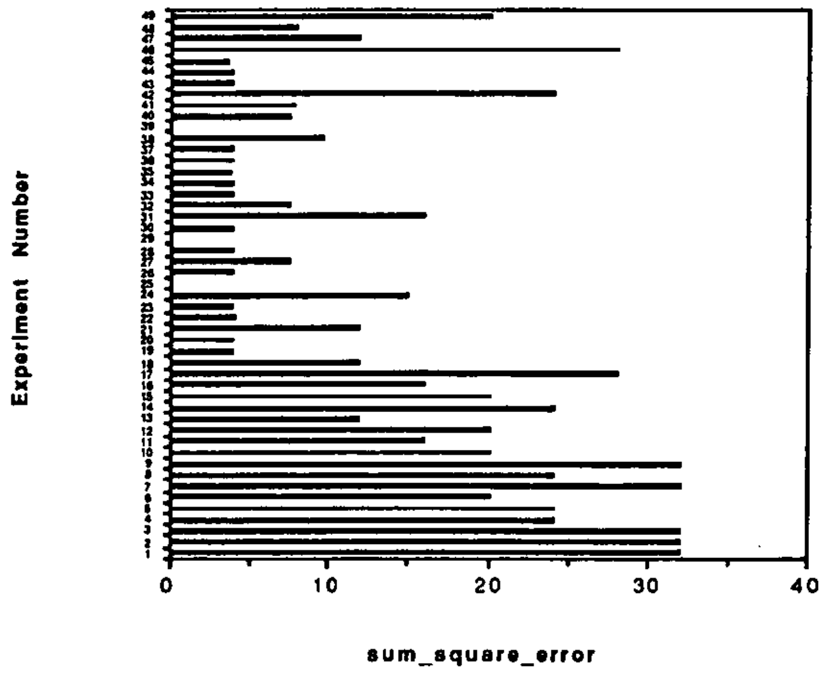
Graph A1.1a. Sum square error performance of training algorithm with three input even parity



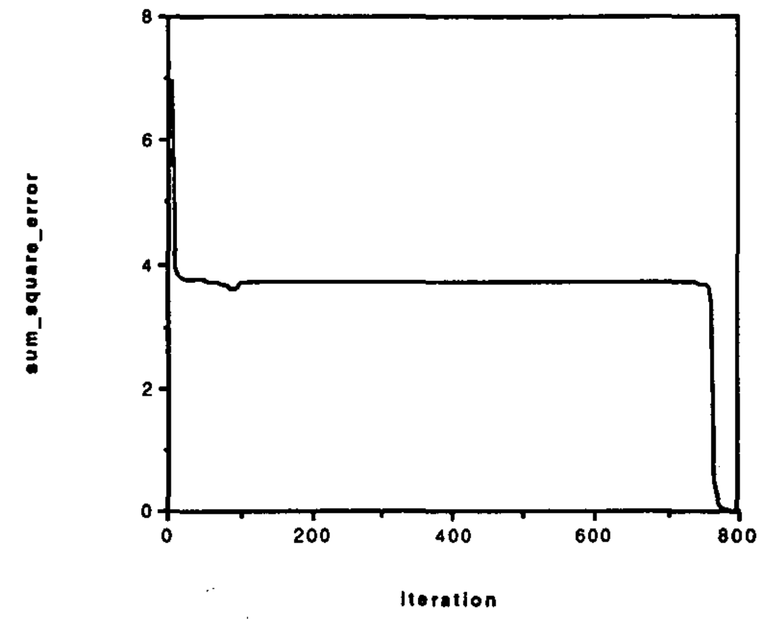
Graph A1.1b. Sum square error performance of training algorithm with three input odd parity



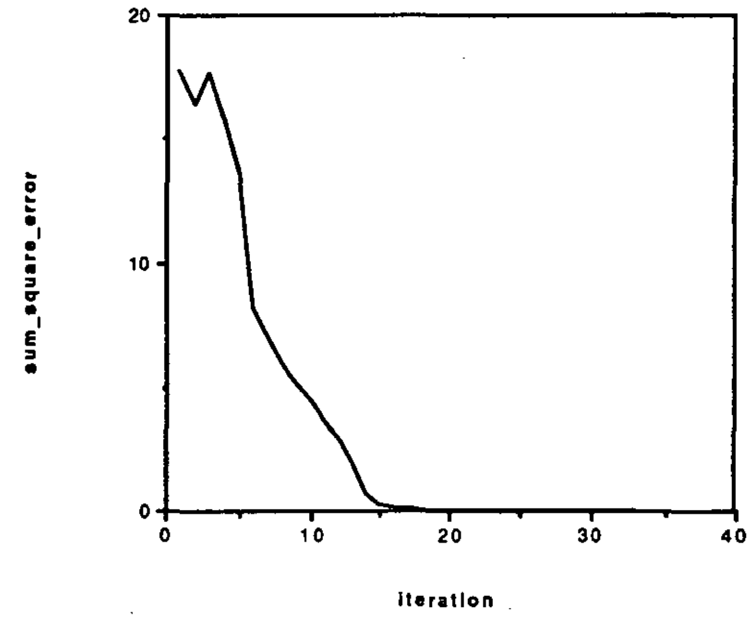
Graph A1.1c. Sum square error performance of training algorithm with four input even parity



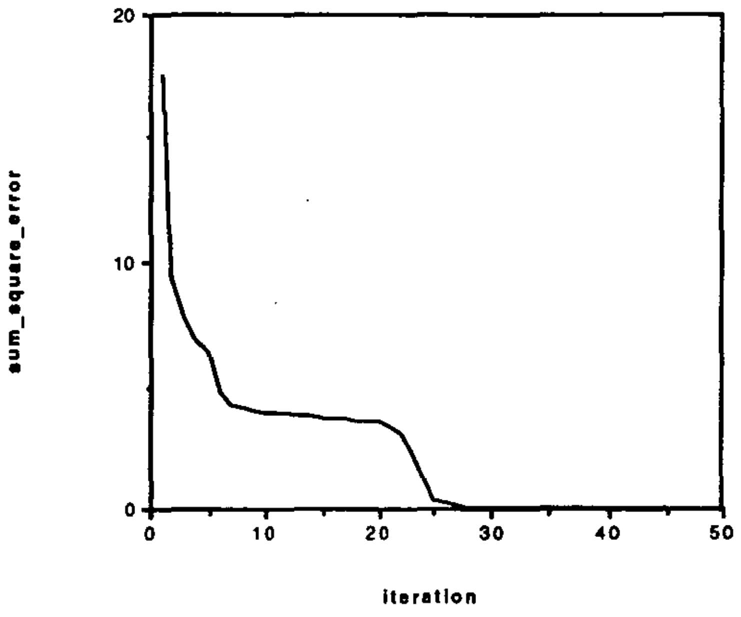
Graph A1.1d. Sum square error performance of training algorithm with four input odd parity



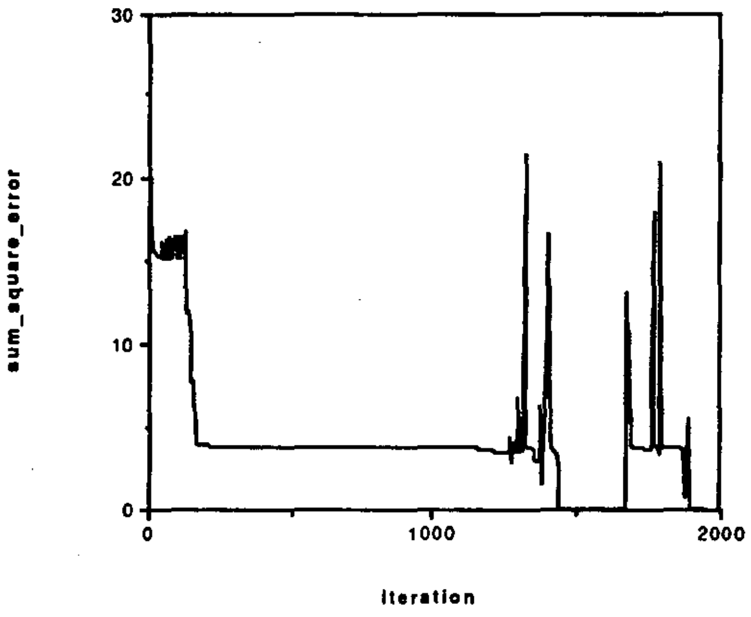
Graph A1.2 a. Sum squared error performance of a specific neural network start point trained on three input even parity



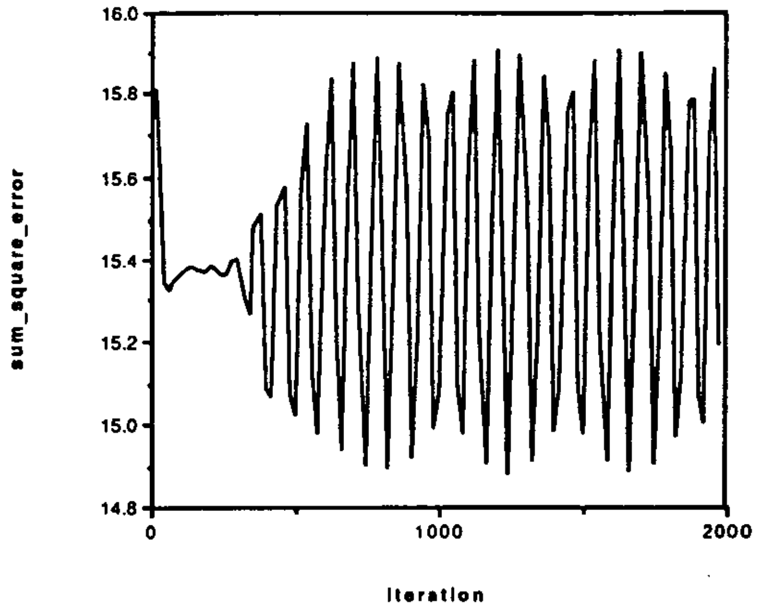
Graph A1.2 b. Sum squared error performance of a specific neural network start point trained on three input odd parity



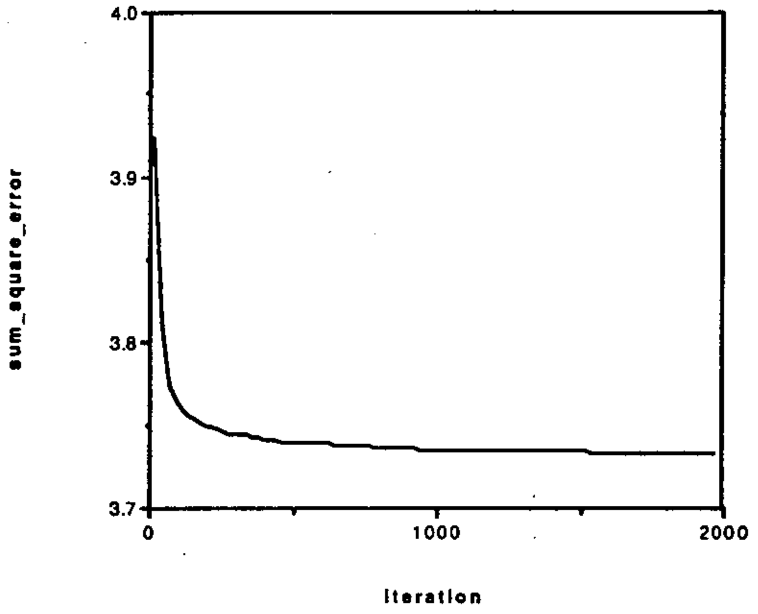
Graph A1.2 c. Sum squared error performance of a specific neural network start point trained on three input even parity



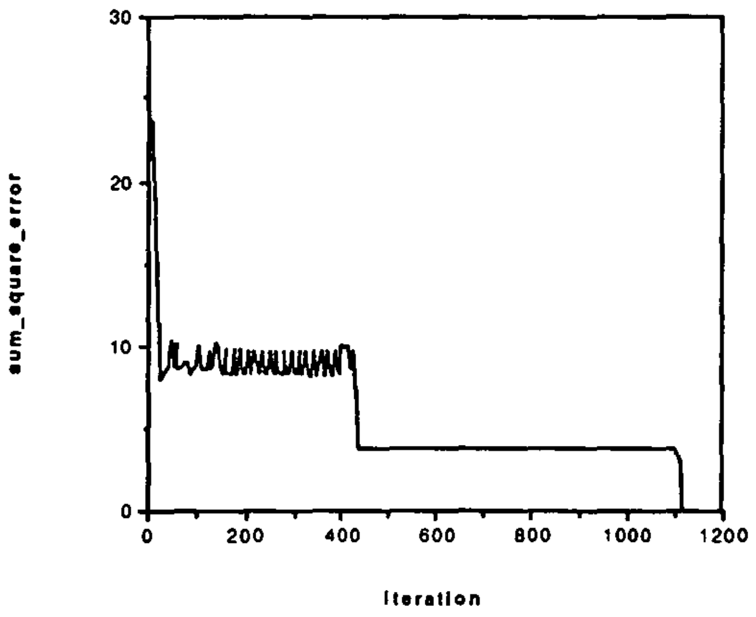
Graph A1.2 d. Sum squared error performance of a specific neural network start point trained on four input even parity



Graph A1.2 e. Sum squared error performance of a specific neural network start point trained on four input odd parity



Graph A1.2 f. Sum squared error performance of a specific neural network start point trained on four input even parity



Graph A1.2 g. Sum squared error performance of a specific neural network start point trained on four input odd parity

Appendix A2. Experiment on Learning Rate

Aim:

The effect of various different learning rates, that is the constant μ in the update formula of the backpropagation algorithm;

$\Delta w = -\mu \, \partial E/\partial w$,

on convergence performance is examined.

Method:

The four data sets which where selected to examine the performance of the backpropagation training algorithm over different learning rates where odd and even parity with three and four inputs. These data sets are as defined in Appendix A1.

The initial network structures that were employed were selected by examining the results of the experiment in appendix A1. An initial network structure was selected that provided convergence before 2000 iterations at the temperature of 0.1 and the learning rate of 0.01. These initial nets where;

for the three input even parity case, net generated with seed 31.

for the three input odd parity case, net generated with seed 52.

for the four input even parity case, net generated with seed 32.

for the four input odd parity case, net generated with seed 38.

The convergence of the four initial network configurations where examined over 3000 iterations at the temperature of 0.1 for various different learning rate values. The results are presented in graphs A2.1 a- d.

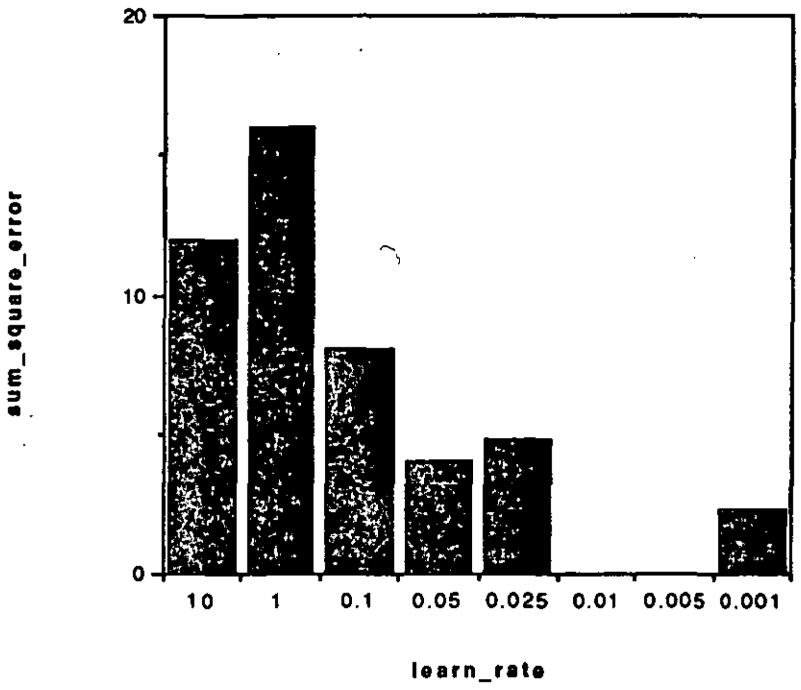
Results:

The graphs A.2.1 a-d show that for low learning rates, learning rate < 0.001, and high learning rate, learning rate > 0.05, the training had not converged in the 3000 iterations examined. For the examples examined in this experiment, the ideal learning rate was seen to be 0.01.

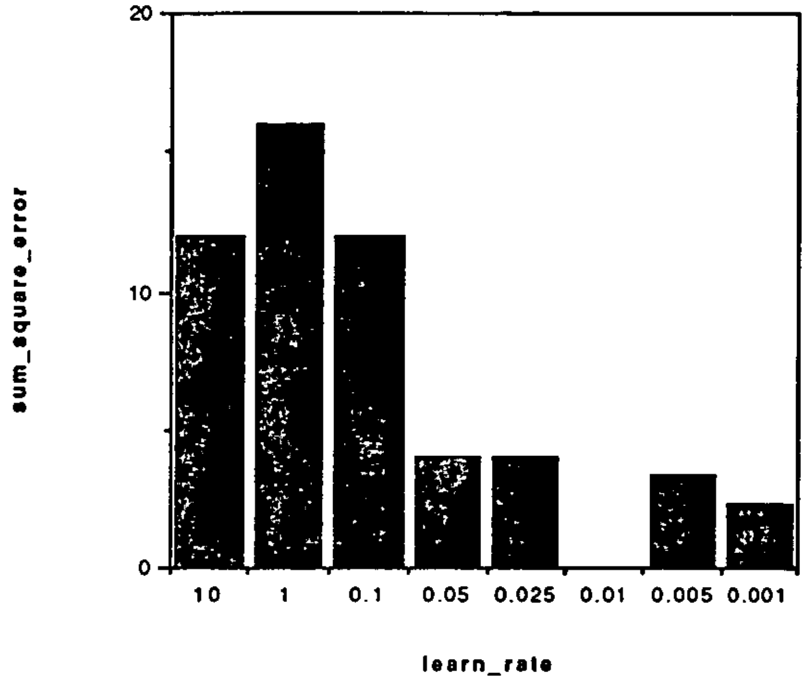
Analysis:

When the learning rate is large the weight update values are large and so gradient descent does not occur. The updates essentially over shoot the ideal path. When the learning rate is very low, the weight updates become very small, true gradient descent occurs. However, two problems exist for this case. Firstly, the updates are so small many iterations are required for the algorithm to converge. Secondly, the algorithm may get stranded in local minima, since the updates are so small, when a local minima is traversed the algorithm, may get stranded.

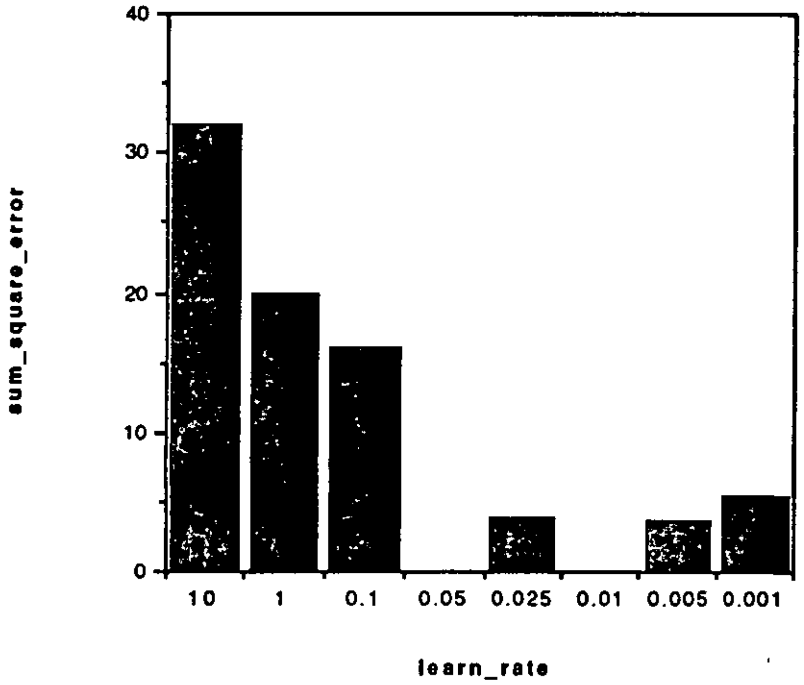
Graphs:



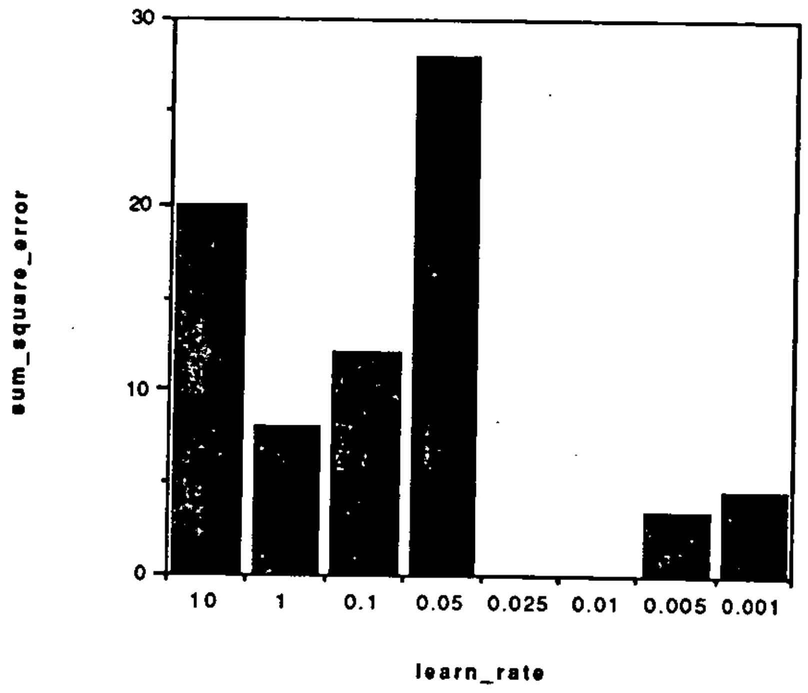
Graph A2.1 a. Sum squared error performance of training algorithm with three input even parity given different learning rates



Graph A2.1 b. Sum squared error performance of training algorithm with three input odd parity given different learning rates



Graph A2.1 c. Sum squared error performance of training algorithm with four input even parity given different learning rates



Graph A2.1 d. Sum squared error performance of training algorithm with four input odd parity given different learning rates

Appendix A3. Experiment on Temperature Value

Aim:

The effect of various different temperature values on the backpropagation algorithms convergence performance is examined. The temperature value is the constant T in the node formula;

Output = (2/(1 + exp(-Weighted_Input/T)) - 1)

Method:

The four data sets which where selected to examine the performance of the backpropagation training algorithm over different learning rates where odd and even parity with three and four inputs. These data sets are as defined in Appendix A1.

The initial network structures that were employed were identical to those discussed in appendix A2. The convergence of the four initial network configurations where examined over 3000 iterations at the learning rate of 0.01 for various different temperature values. The results are presented in graphs A3.1 a- d.

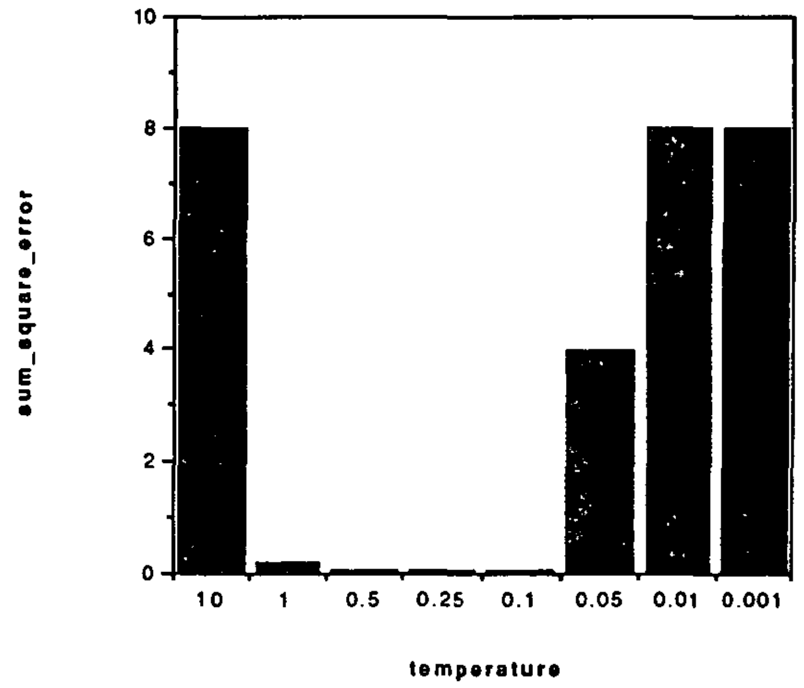
Results:

The graphs A.3.1 a- d show that for low temperature values, temperature < 0.05, and high temperature values, temperature > 0.5, the training had not converged in the 3000 iterations examined. For the examples examined in this experiment, the ideal temperature value was seen to be 0.1.

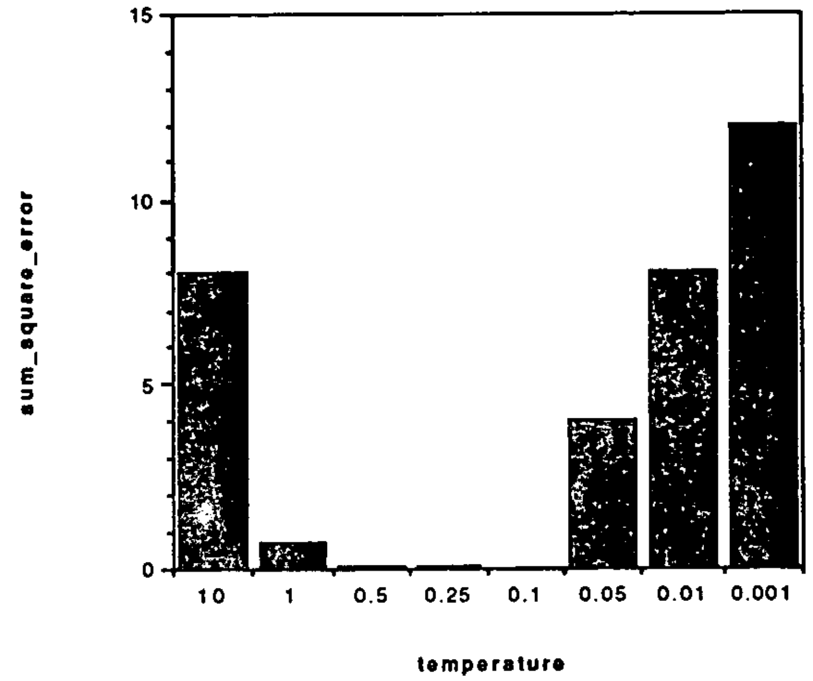
Analysis:

When the temperature is large the sigmoid function is soft, that is the derivative of the sigmoid function is not large. This means that the update values of the weights remain small. Also since the soft sigmoid function requires large inputs to provide outputs in the region ±1 training must progress for a long time before convergence occurs. When the temperature value is very low the sigmoid function is hard. That is its derivative is almost zero everywhere except near zero where it is large. If the node inputs are near zero, the problem associated with a high learning rate, that of large weight update occurs. Gradient descent does not occur. If the node inputs are not near zero, the hard sigmoid ensures that the weight updates are very small. This is the same problem as that associated with a low learning rate, the updates are so small that convergence will take many iterations while the chance of stranding in local minima is increased greatly.

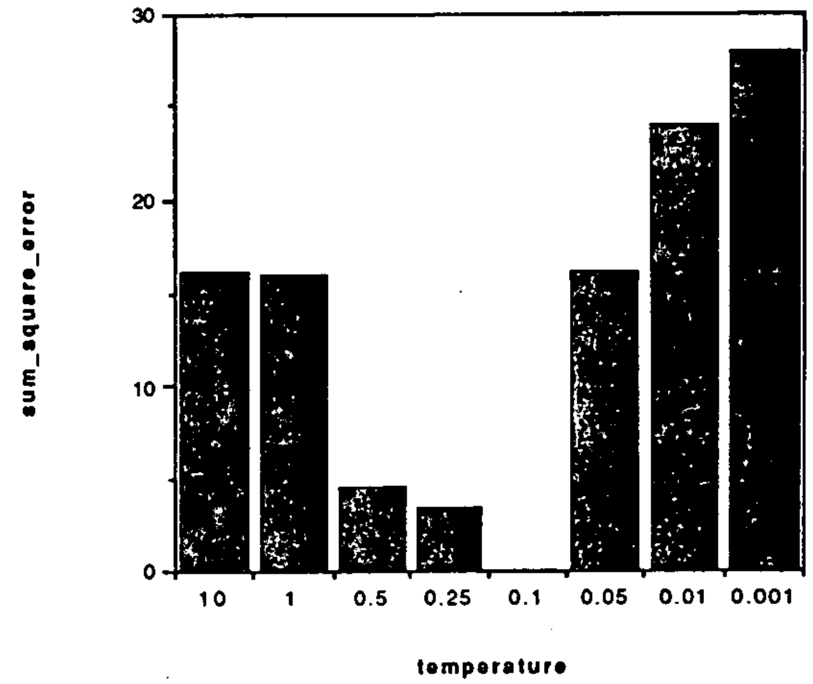
Graphs:



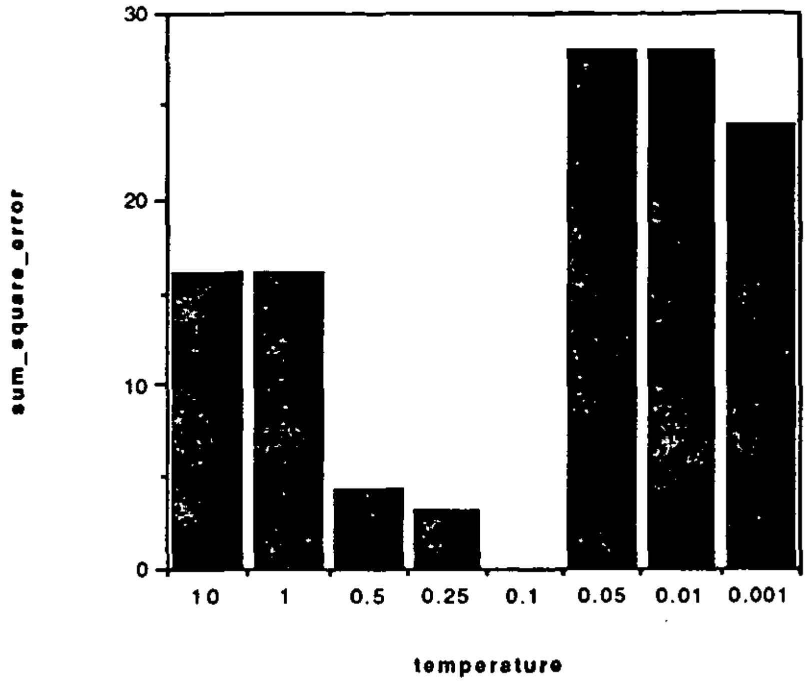
Graph A3.1 a. Sum squared error performance of training algorithm with three input even parity given different temperature values



Graph A3.1 b. Sum squared error performance of training algorithm with three input odd parity given different temperature values



Graph A3.1 c. Sum squared error performance of training algorithm with four input even parity given different temperature values



Graph A3.1 d. Sum squared error performance of training algorithm with four input odd parity given different temperature values

Appendix A4. Retraining Neural Networks on New Training Data

Aim:

To observe the convergence properties of training models of odd and even parity on even and odd training data respectively.

Method:

Models of odd and even parity with three and four inputs were selected from those neural network models that converged in the experiments of appendix A1. These are identical to those discussed in appendix A2 and A3. These neural networks were retrained on opposite data for a maximum of 1000 iterations at a learning rate of 0.01 and a temperature of 0.1. The convergence results are shown in graphs A.4.1 a- d.

Results:

The convergence performance of the four experiments as shown in graphs A.4.1 a- d have the following sum squares errors of;

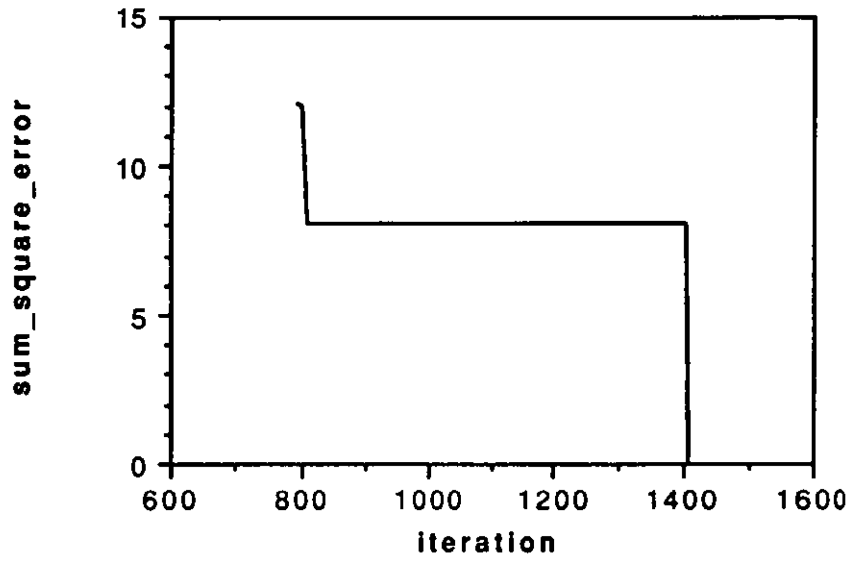
a. 0 b. 0 c. 4 d. 12.

The three input parity experiments converged in under 1000 iterations, this was in 600 and 200 iterations respectively. The four input parity experiments did not converge. Retraining neural network models to new conflicting data is very difficult for large input spaces.

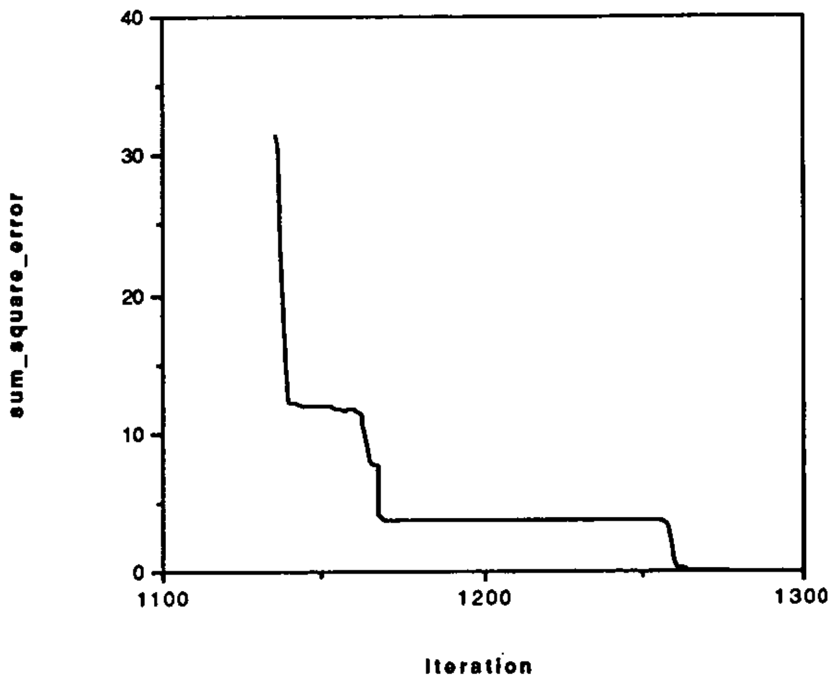
Analysis:

The neural models of odd and even parity differ by a negation of the output weights or the hidden layer weights. That is the update of just one of the layers of weights leading to the negation of all the weight values. However, the update of the training scheme is iterative over the whole weight space and so can not isolate the update in the optimal manner. There is an interfering effect over the updates that are provided by the training algorithm.

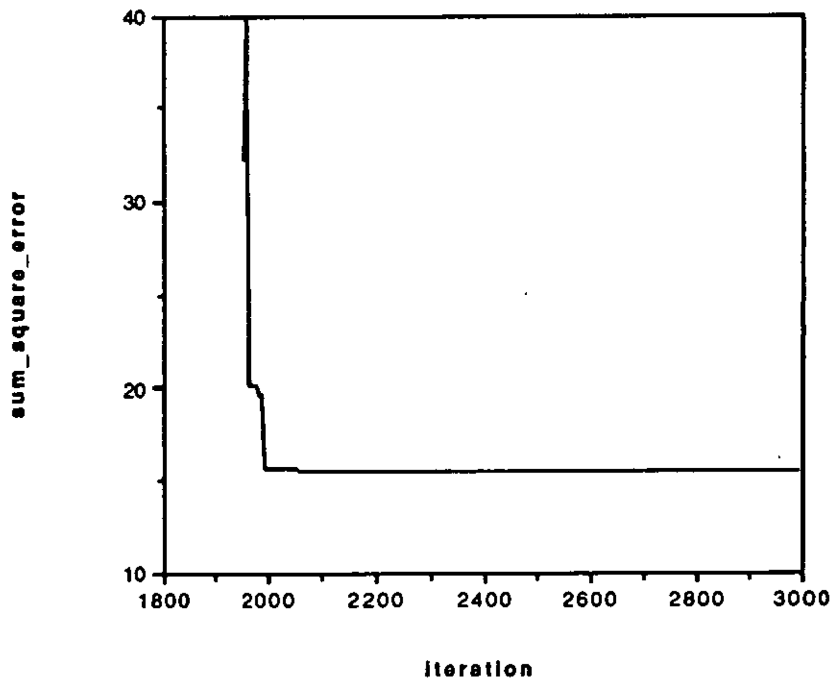
Graphs:



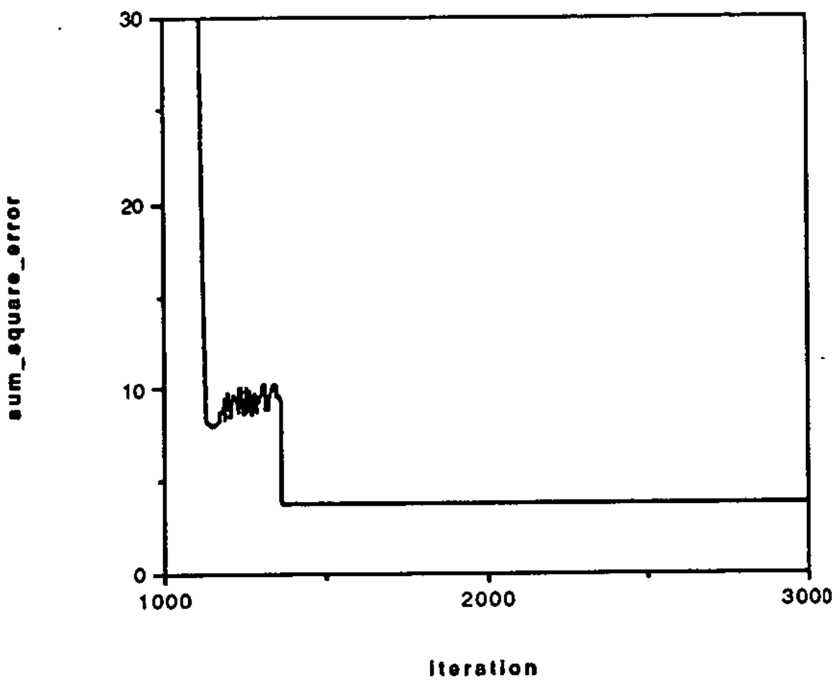
Graph A4.1 a. Sum squared error convergence of neural network modelled on three input even parity , when trained on odd parity



Graph A4.1 b. Sum squared error convergence of neural network modelled on three input odd parity , when trained on even parity



Graph A4.1 c. Sum squared error convergence of neural network modelled on four input even parity , when trained on odd parity



Graph A4.1 d. Sum squared error convergence of neural network modelled on four input odd parity , when trained on even parity

Appendix B. Experiment on Learning Performance

Aim:

The aim of this experiment is to identify the learning performance of the backpropagation training algorithm over different training data and different initial neural network configurations.

Method:

The data that was used for this experiment was generated via a seeded pseudorandom scheme. The output values were generated iteratively for all the possible input values. The first output value is generated from the seed via the formula;

Outputvalue= ((Seed mod 2) - 1/2) * 2,

Nextseed= (Seed * 101 * 992) mod 1009,

and the next output value is generated from the nextseed value. This is continued iteratively for all the input cases. The input values start at (1,1,...,1,1) then (1,1,...,1,-1) and (1,1,...,-1,1) and (1,1,...,-1,-1) and progress iteratively to (-1,-1,...,-1,-1).

The initial network configurations of the neural networks were generated in the manner as discussed in appendix A1. In this case an n input neural network with n hidden nodes and one output nodes is fully defined by $(n + 1)^2$ weight values.

Twenty four experiments were conducted for each number of input nodes. These experiments were generated by selecting a pseudorandom data set and a pseudorandom initial neural network configuration given by the formula;

Dataseed= 7 * Inputnodes,

Netseed= 3 * Inputnodes,

Experiment is net(Netseed) trained on data(Dataseed),

Nextdataseed= (Dataseed * 101 * 992) mod 1009, Nextnetseed= (Netseed * 101 * 992) mod 1009.

The neural networks were trained over 1000 iterations at a temperature of 0.1 and a learning rate of 0.01. The results are presented in graphs B.1 a- d and B.2 a- d. Several typical sum squares error convergence characteristics are shown in fig B.3 a- o.

Results:

The low input node experiments in general converged before the 1000 iteration mark, and so the number of iterations that were required for convergence are a good measure of convergence performance. The experiments with many input nodes generally did not converge to the optimal solution within the 1000 iterations and so the sum squared error value after 1000 iterations is a good measure of convergence performance. These figures are shown in table B.1.

Two input case;	19/ 24 experiments converged within 100 iterations.
Three input case;	12/ 24 experiments converged within 100 iterations.
Four input case;	13/ 24 experiments converged within 100 iterations.
Five input case;	7/ 24 experiments converged within 200 iterations.

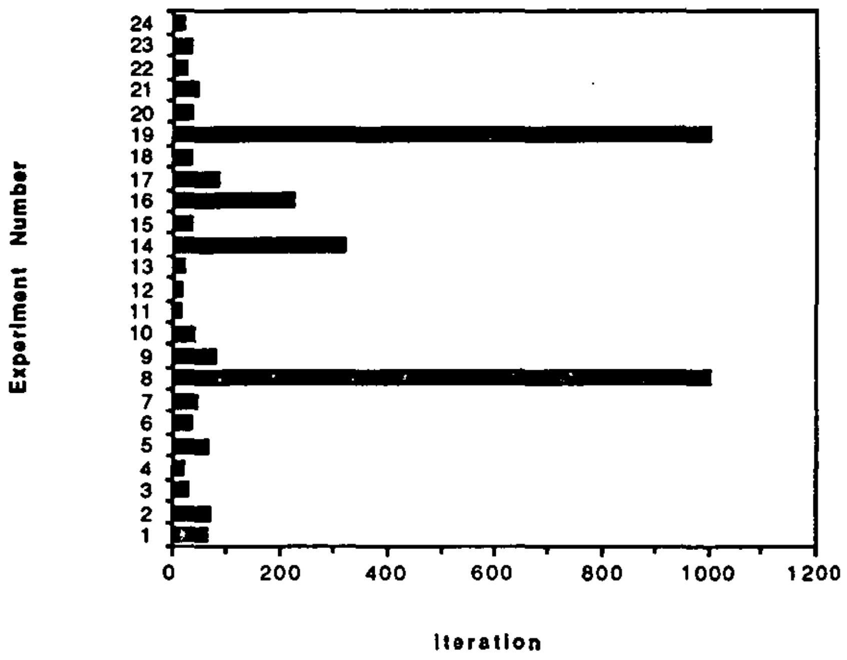
Experiment	Mean	Variance
2 input case	0.319	0.999
3 input case	0.943	1.587
4 input case	0.934	1.562
5 input case	3.066	3.562
6 input case	5.926	4.509
7 input case	27.185	7.480
8 input case	151.930	47.434

Table B.1. Mean and variance of the sum squared error performance after 1000 iterations

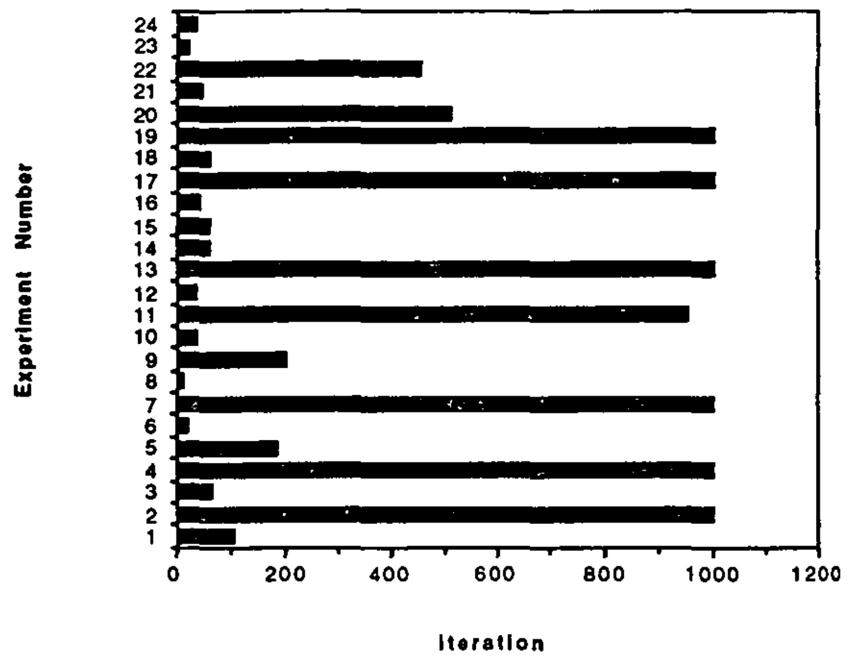
Analysis:

These results show that the backpropagation algorithm converges quickly for a small number of inputs. As the number of inputs increases the convergence performance deteriorates quickly. These results serve as a good performance measure against other neural network structures such as the Ghost node neural networks discussed in appendix C.

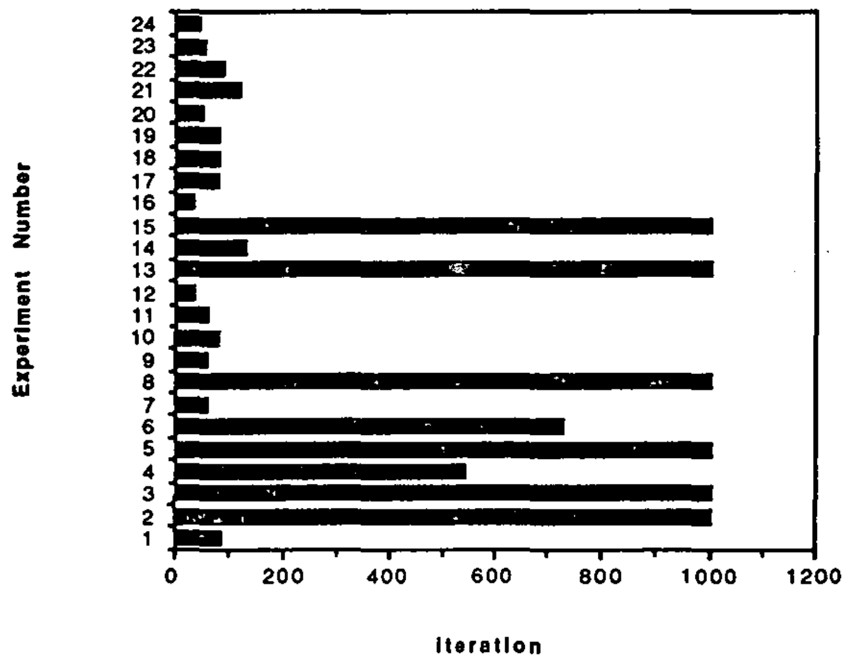
Graphs:



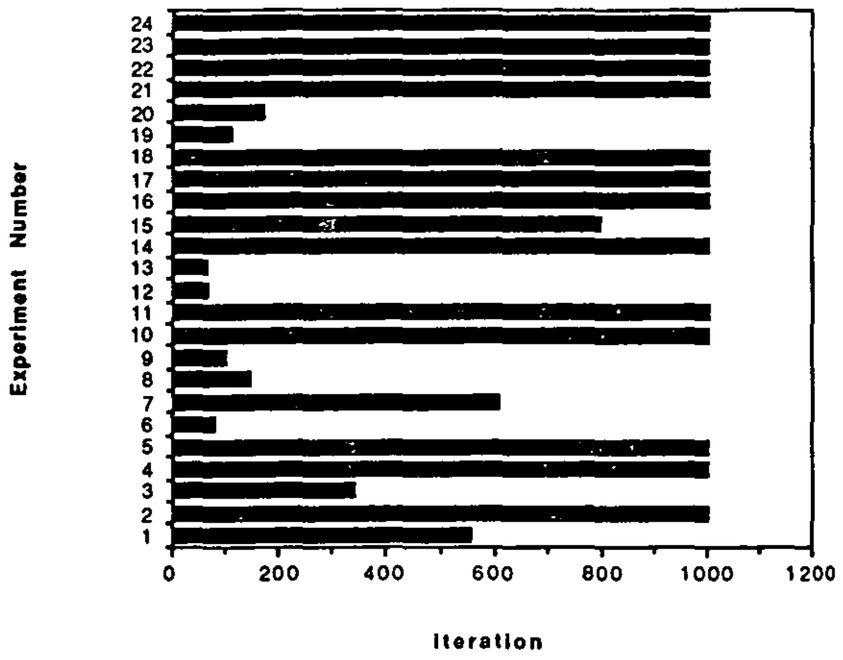
Graph B.1 a. Number of iterations required for convergence of two input neural networks (maximum number of iterations allowed is 1000)



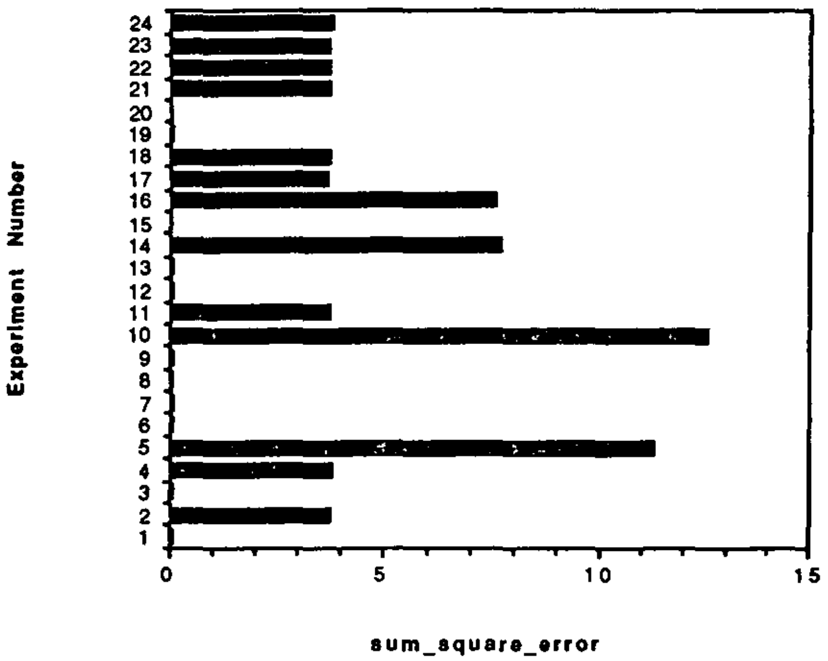
Graph B.1 b. Number of iterations required for convergence of three input neural networks (maximum number of iterations allowed is 1000)



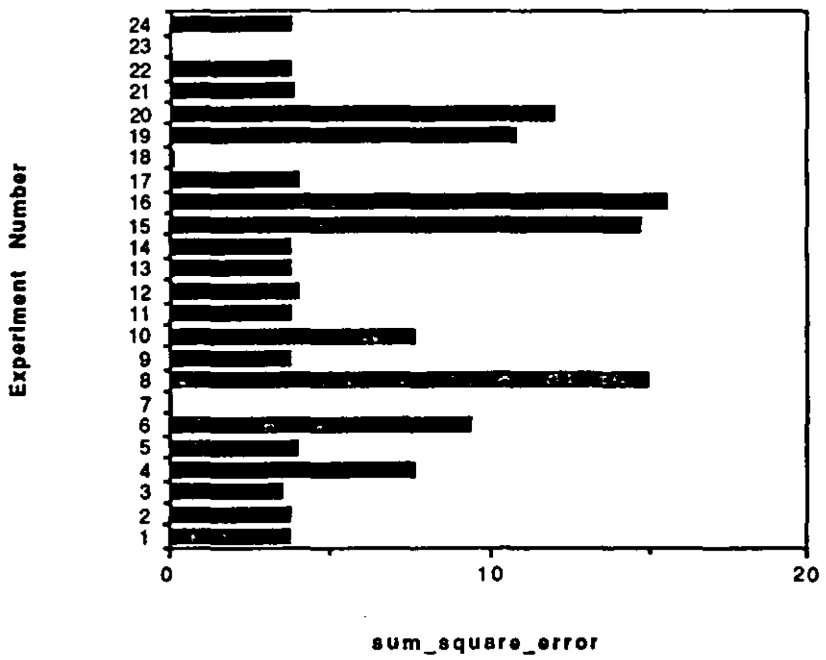
Graph B.1 c. Number of iterations required for convergence of four input neural networks (maximum number of iterations allowed is 1000)



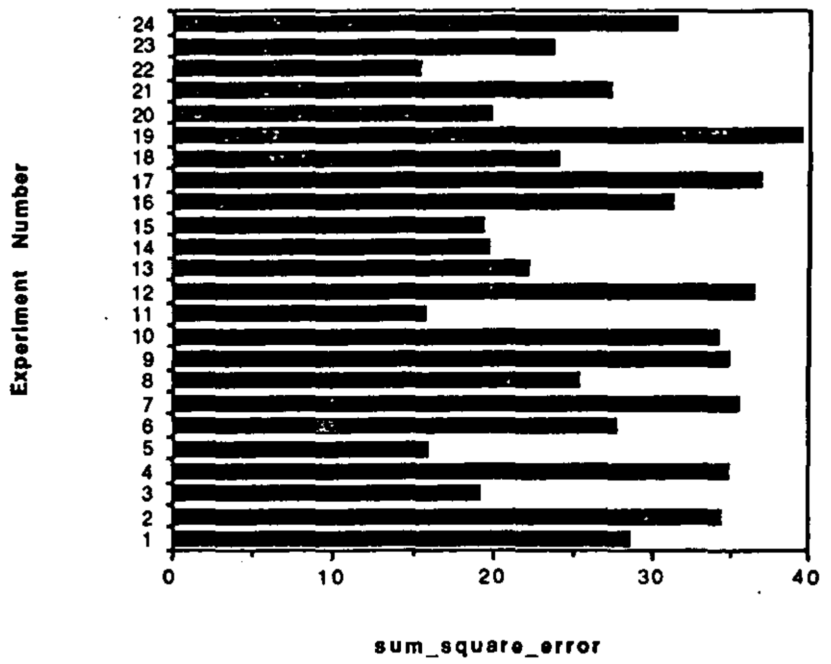
Graph B.1 d. Number of iterations required for convergence of five input neural networks (maximum number of iterations allowed is 1000)



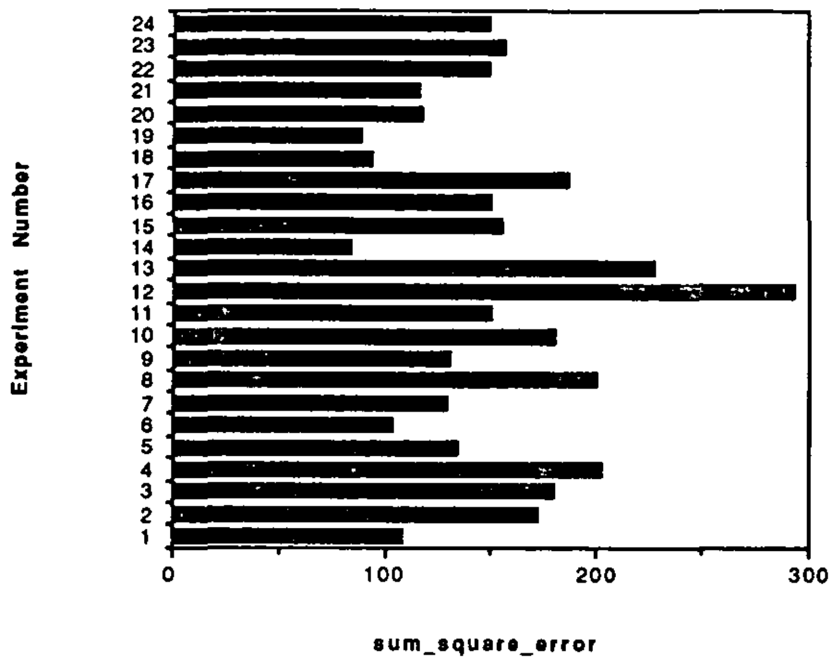
Graph B.2 a. Sum squared error values of five input neural networks after 1000 iterations



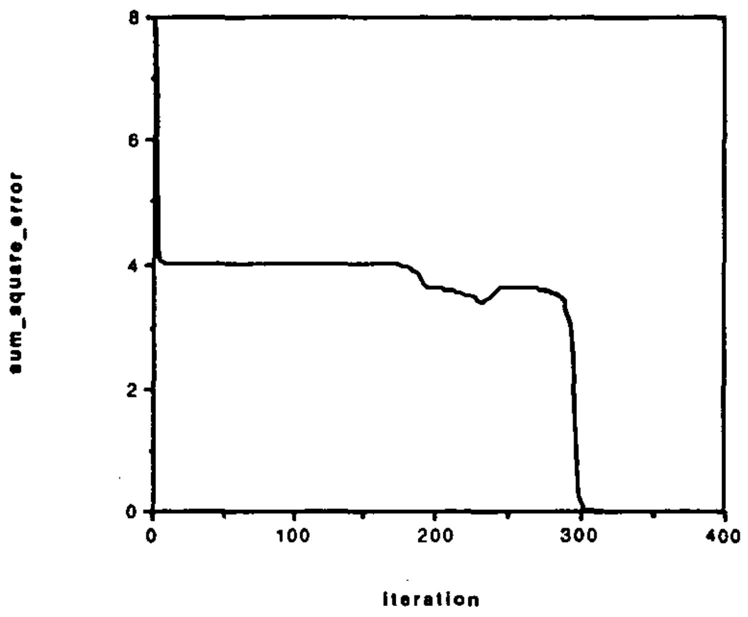
Graph B.2 b. Sum squared error values of six input neural networks after 1000 iterations



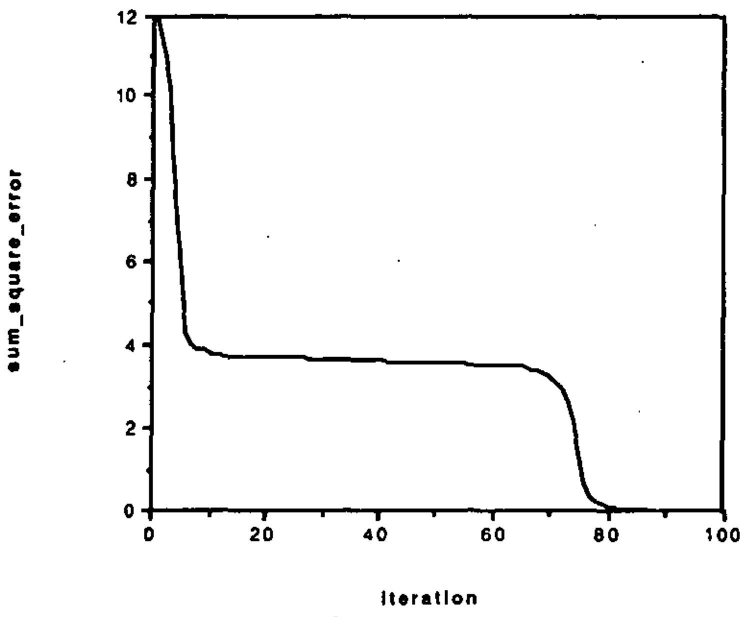
Graph B.2 c. Sum squared error values of seven input neural networks after 1000 iterations



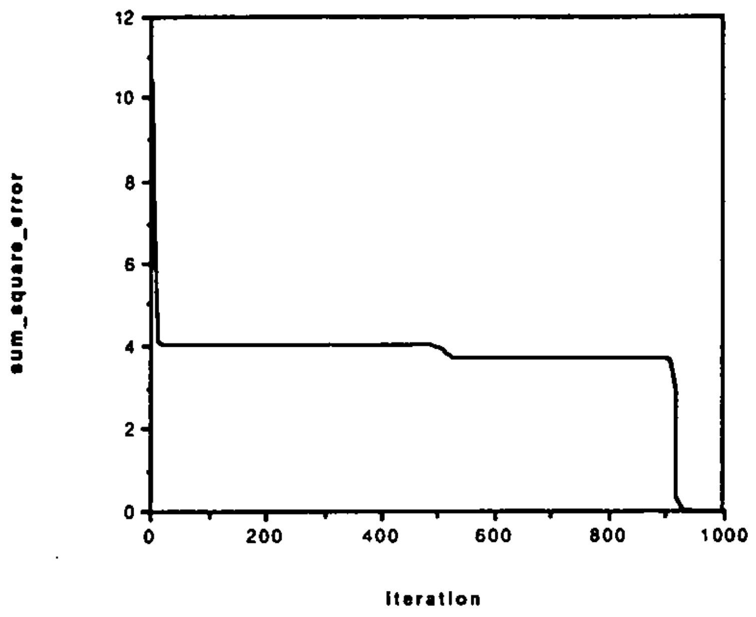
Graph B.2 d. Sum squared error values of eight input neural networks after 1000 iterations



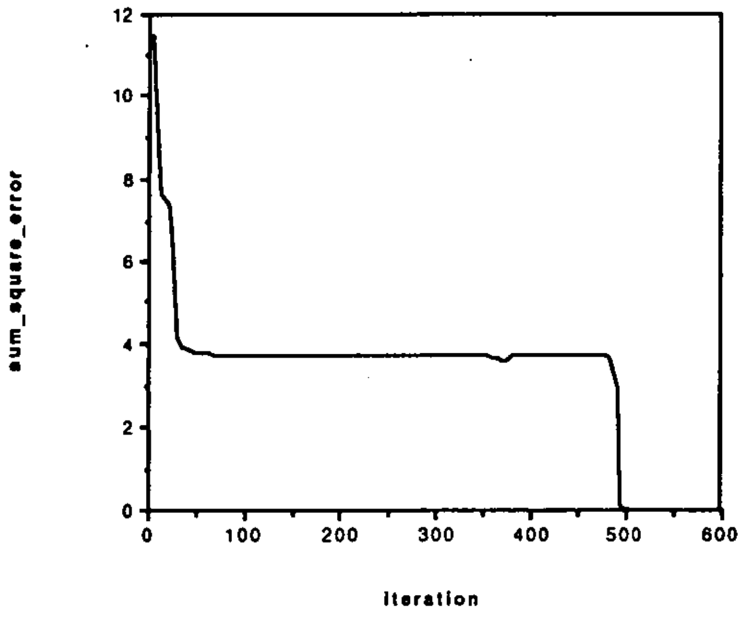
Graph B.3 a. Sum squared error convergence performance of a two input neural network with a specific training set



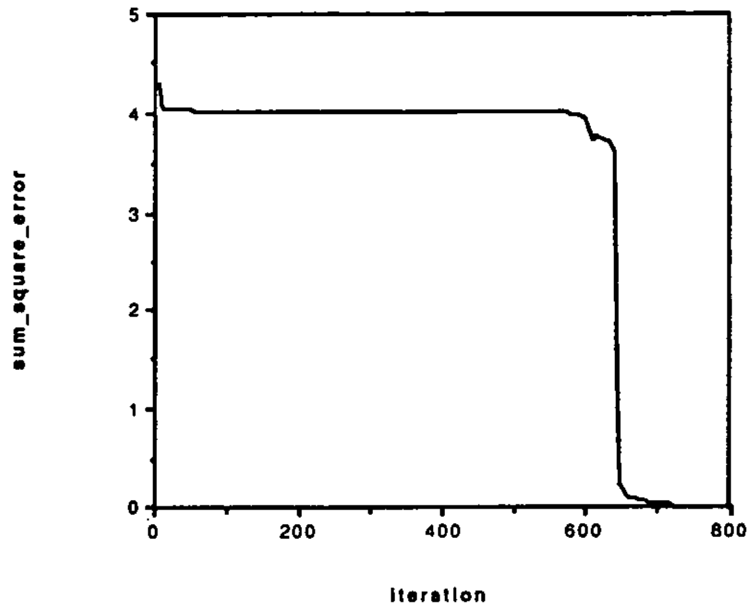
Graph B.3 b. Sum squared error convergence performance of a two input neural network with a specific training set



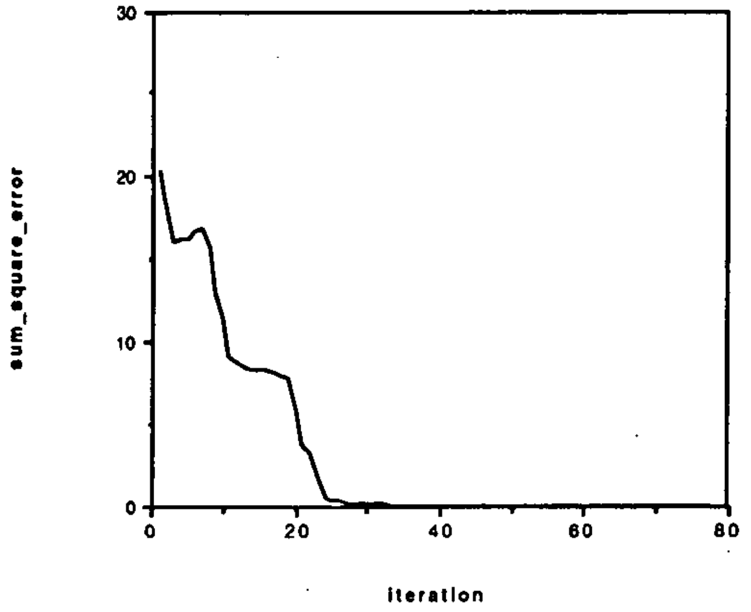
Graph B.3 c. Sum squared error convergence performance of a three input neural network with a specific training set



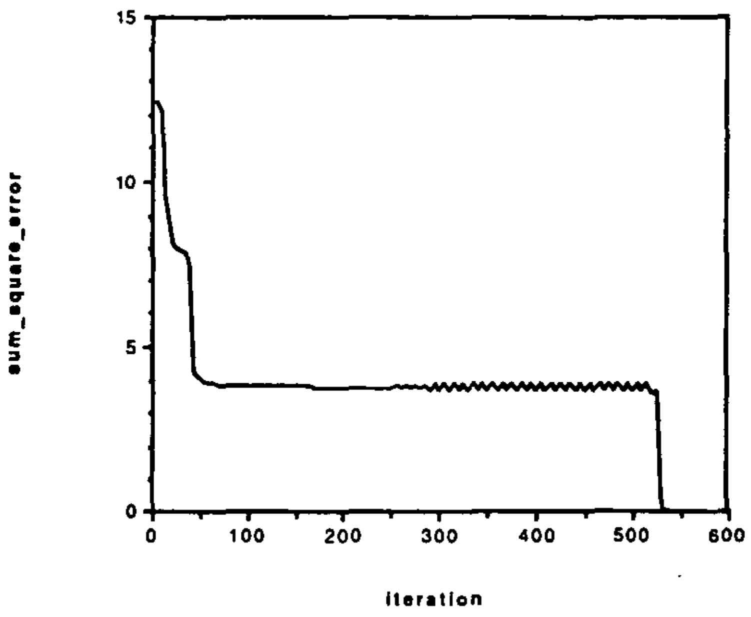
Graph B.3 d. Sum squared error convergence performance of a three input neural network with a specific training set



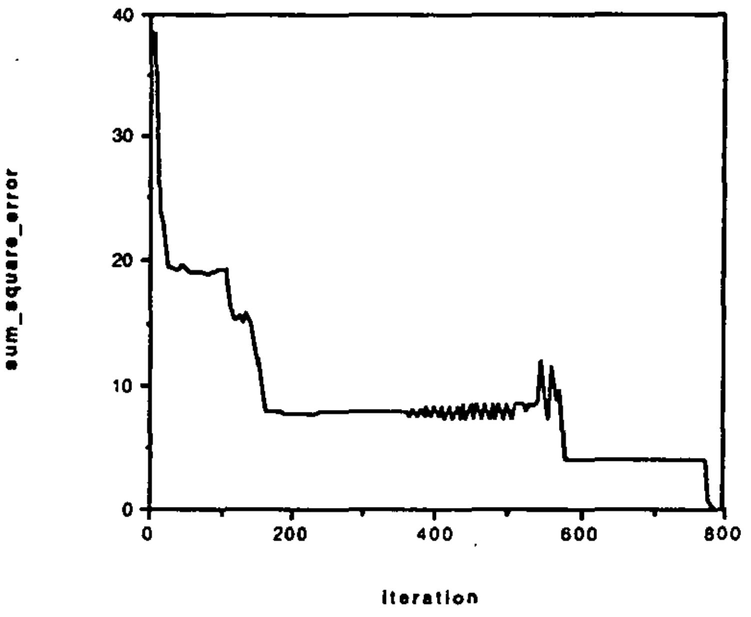
Graph B.3 e. Sum squared error convergence performance of a four input neural network with a specific training set



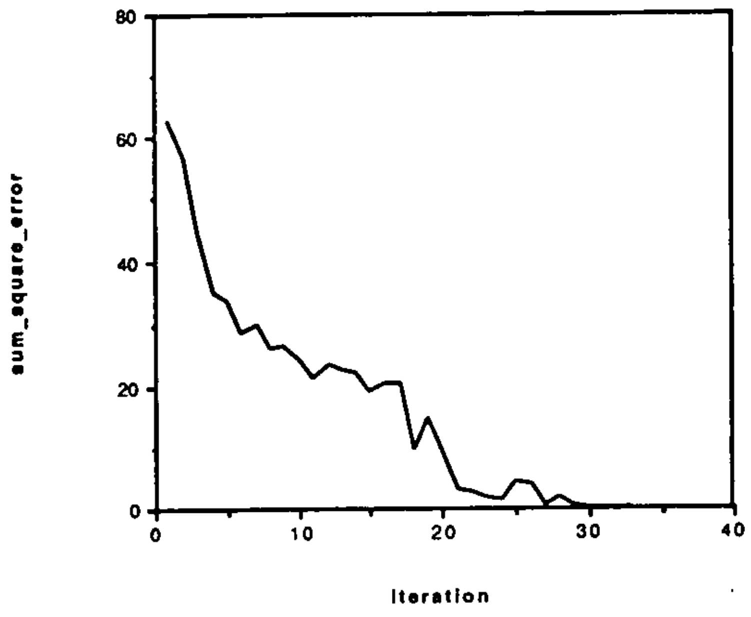
Graph B.3 f. Sum squared error convergence performance of a four input neural network with a specific training set



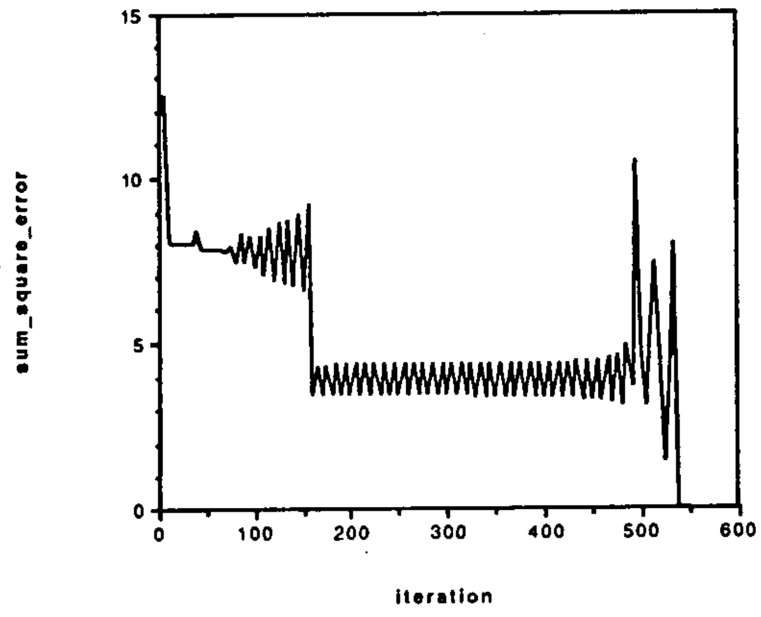
Graph B.3 g. Sum squared error convergence performance of a four input neural network with a specific training set



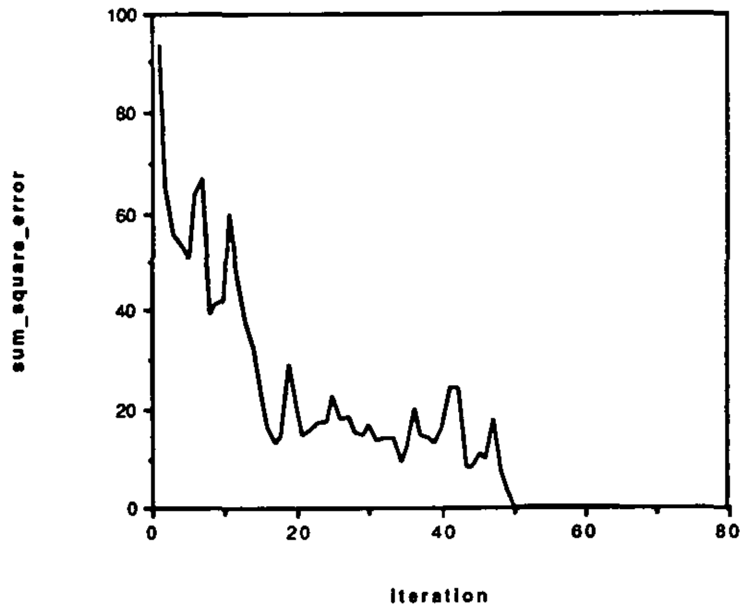
Graph B.3 h. Sum squared error convergence performance of a five input neural network with a specific training set



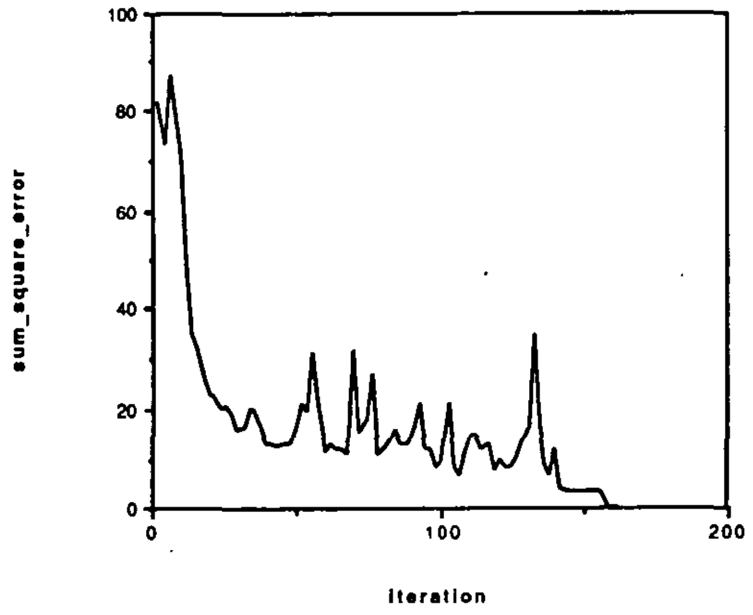
Graph B.3 i. Sum squared error convergence performance of a five input neural network with a specific training set



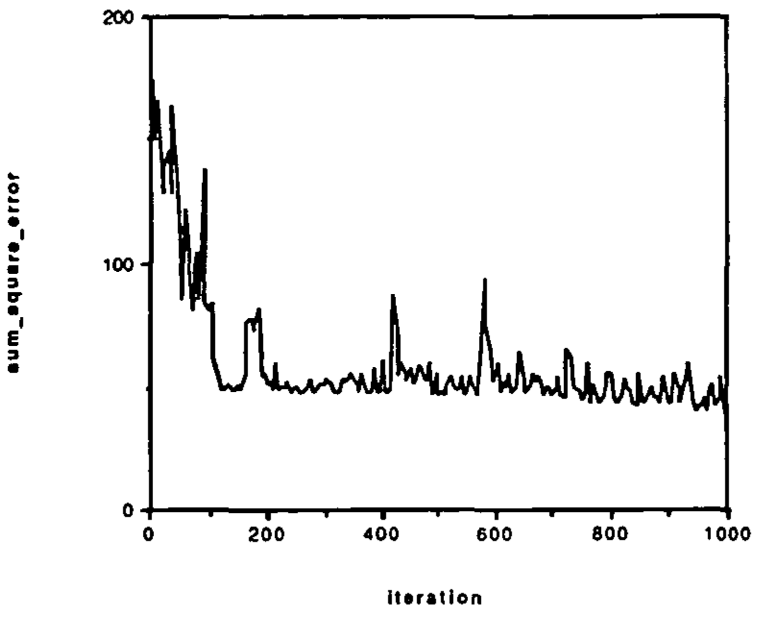
Graph B.3 j. Sum squared error convergence performance of a five input neural network with a specific training set



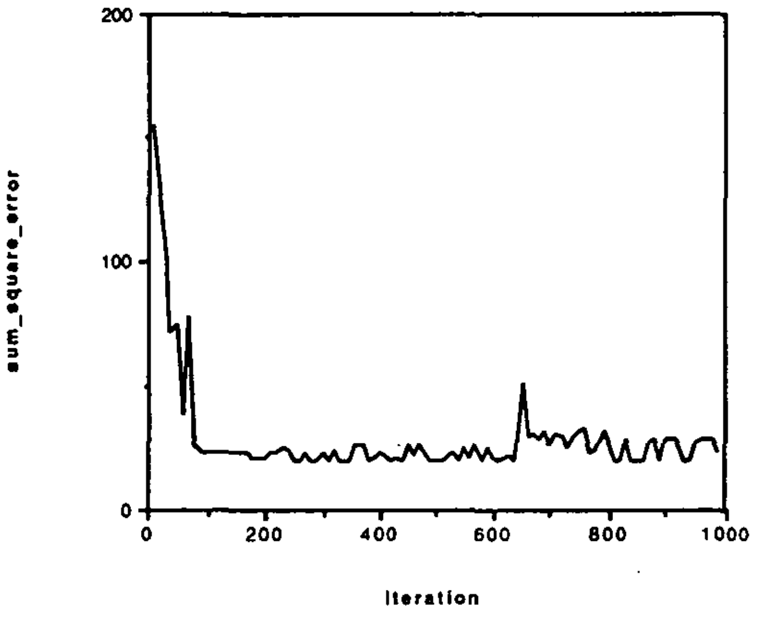
Graph B.3 k. Sum squared error convergence performance of a six input neural network with a specific training set



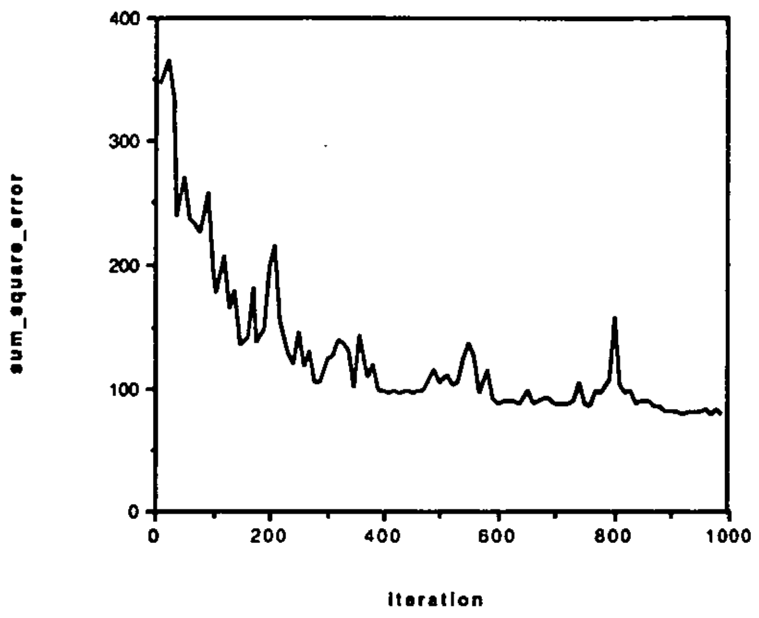
Graph B.3 l. Sum squared error convergence performance of a six input neural network with a specific training set



Graph B.3 m. Sum squared error convergence performance of a seven input neural network with a specific training set



Graph B.3 n. Sum squared error convergence performance of a seven input neural network with a specific training set



Graph B.3 o. Sum squared error convergence performance of a eight input neural network with a specific training set

Appendix C. Experiment on Ghost Learning Performance

Aim:

The aim of this experiment is to identify the learning performance of the backpropagation training algorithm over different training data and different initial ghost neural network configurations.

Method:

The data that was used for this experiment was generated in the same manner as the data discussed in appendix B.

The initial network configurations of the neural networks were generated by a pseudorandom scheme similar to that discussed in appendix A1. The ghost node networks have fewer weight values than standard neural network structures. The hidden nodes in the ghost node neural networks share weight values and have distinct biases. Therefore for a fully connected n input ghost node neural network with n hidden ghost nodes and one output node we require 3*n +1 weights. The weights were selected such that the weight values of the ghost nodes were identical to the weight values of the first node in the corresponding neural network in appendix B. The ghost nodes in the hidden layer had biases identical to the biases of the nodes in the network of the corresponding experiment in appendix B.

Twenty four experiments were conducted for each number of input nodes. These experiments were generated by selecting a pseudorandom data set and a pseudorandom initial neural network configuration in an identical manner as to that discussed in appendix B. The neural networks were trained over 1000 iterations at a temperature of 0.1 and a learning rate of 0.01. The results are presented in graphs C.1 a- d, C.2 a- e and table C.1. Several typical sum squares error convergence characteristics are shown in graphs C.3 a- i.

Results:

The low input node experiments in general converged before the 1000 iterations, and so the number of iterations that were required for convergence are a good measure of convergence performance. The experiments with many input nodes generally did not converge to the optimal solution within the 1000 iterations and so the sum squared error value after 1000 iterations is a good measure of convergence performance. The iterative performance of the experiments are shown in graphs C.1 a- d. The sum squared error convergence performance are presented in, C.2 a- e, and table C.1.

Two input case;	18/ 24 experiments converged within 100 iterations.
Three input case;	9/ 24 experiments converged within 100 iterations.
Four input case;	6/ 24 experiments converged within 100 iterations.
Five input case;	4/ 24 experiments converged within 200 iterations.

Experiment	Mean	Variance
2 input case	0.767	1.462
3 input case	1.938	2.542
4 input case	2.003	2.903
5 input case	6.647	5.416
6 input case	25.339	10.687
7 input case	86.902	26.062
8 input case	350.968	57.593

Table C.1. Mean and variance of the sum squared error after 1000 iterations

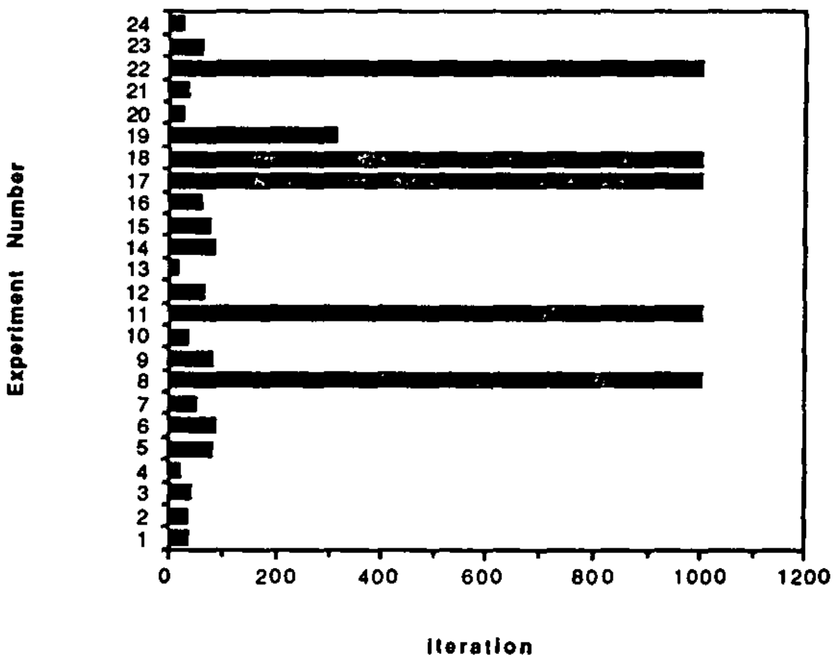
Analysis:

Similar to the standard neural network performance measures these results show that the backpropagation algorithm converges quickly for ghost node neural networks with a small number of inputs. As the number of inputs increases the convergence performance deteriorates quickly.

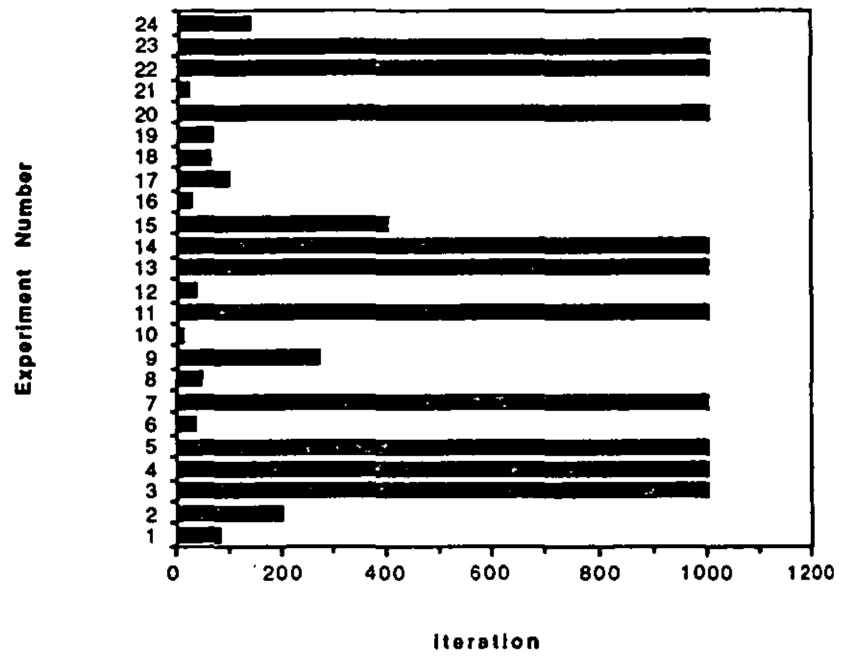
Comparing these results with the results that are given in appendix B, the performance of the ghost node neural networks under backpropagation learning are generally much worse than that of the standard system. The performance over two and three input nodes are comparable. For the larger number input values the sum square error performance of the ghost node neural networks is about twice that of the standard system.

Many isolated examples can be seen where the performance of a ghost node neural network is better than that of the corresponding standard neural network start point. This highlights the fact that the ghost node structure can often better model the transformation in question.

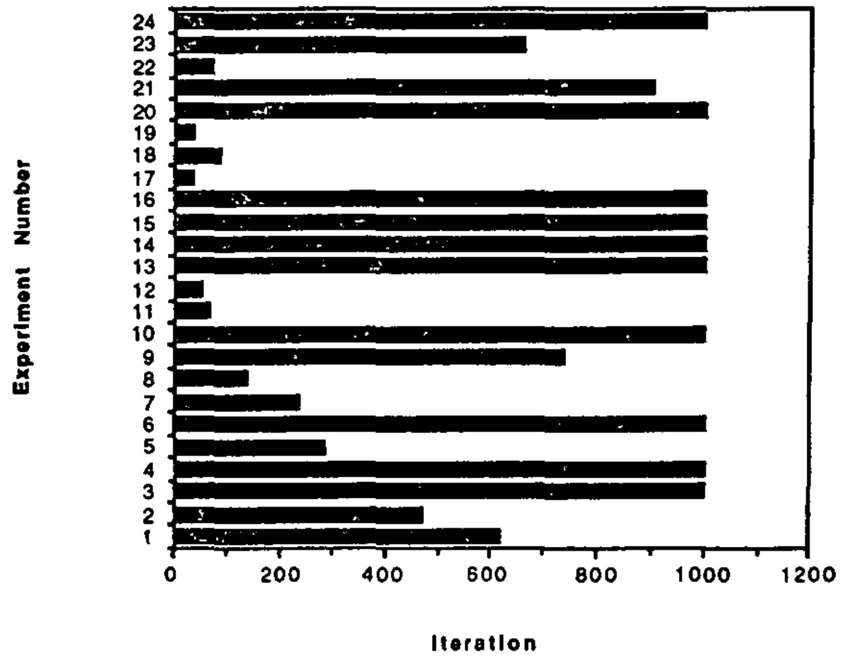
Graphs:



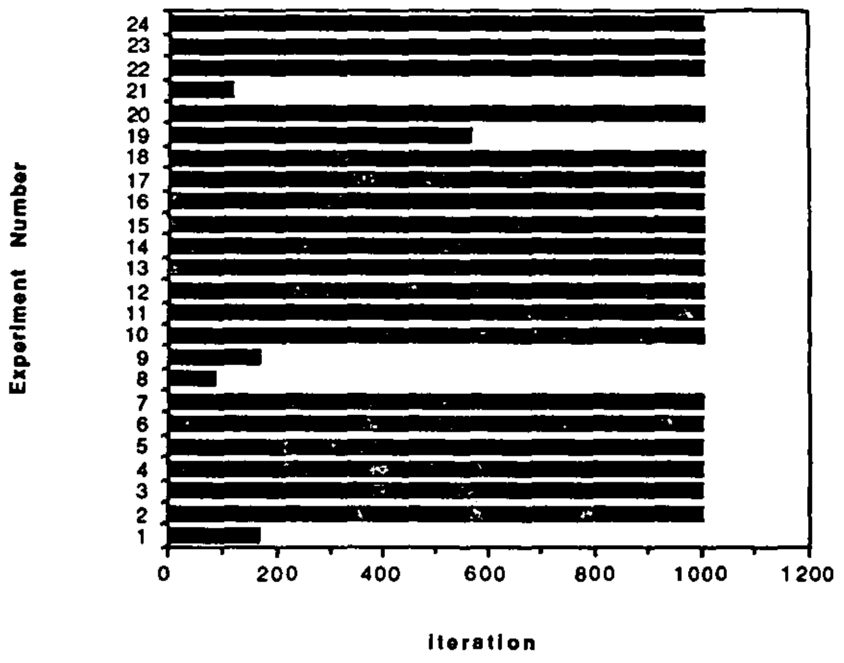
Graph C.1 a. Number of iterations required for convergence of two input ghosted neural networks (maximum number of iterations allowed is 1000)



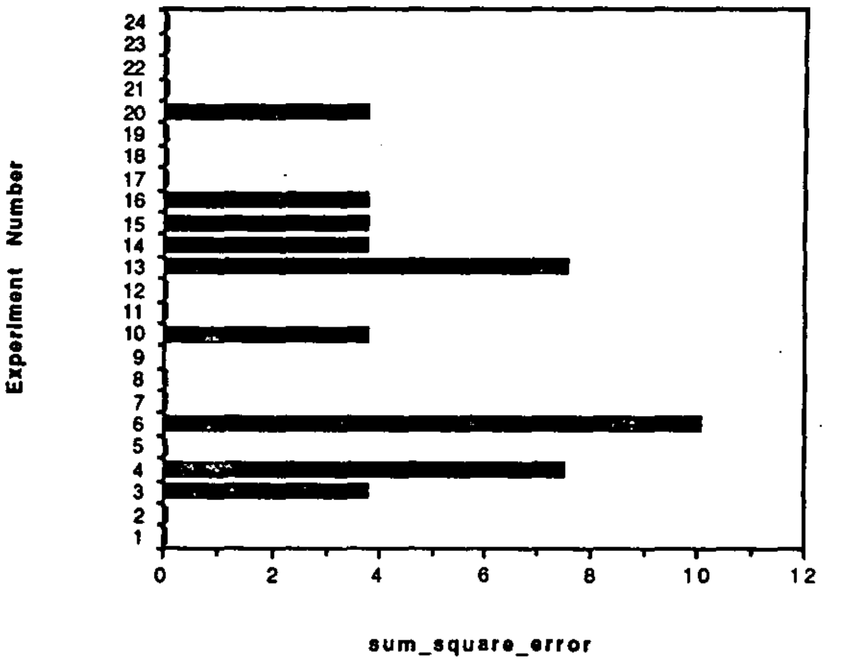
Graph C.1 b. Number of iterations required for convergence of three input ghosted neural networks (maximum number of iterations allowed is 1000)



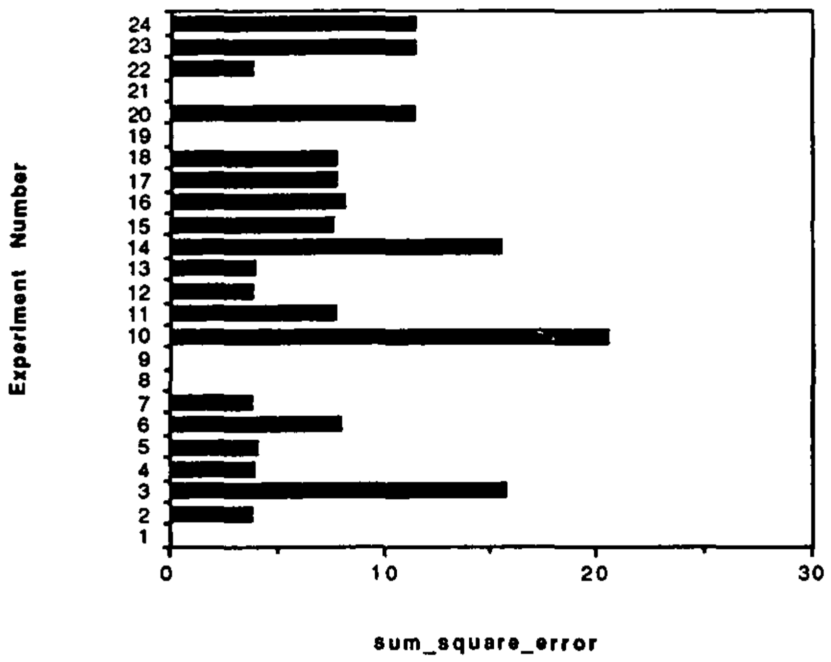
Graph C.1 c. Number of iterations required for convergence of four input ghosted neural networks (maximum number of iterations allowed is 1000)



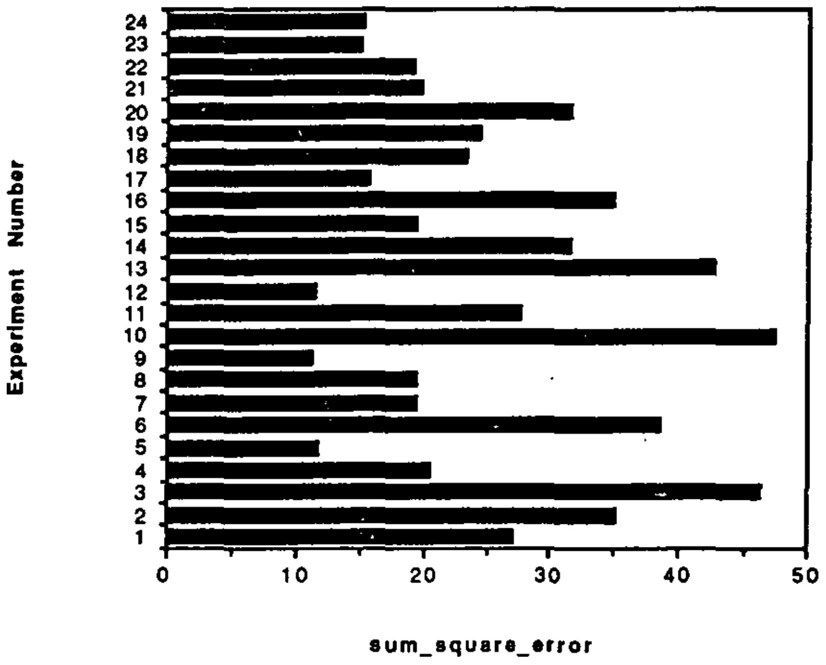
Graph C.1 d. Number of iterations required for convergence of five input ghosted neural networks (maximum number of iterations allowed is 1000)



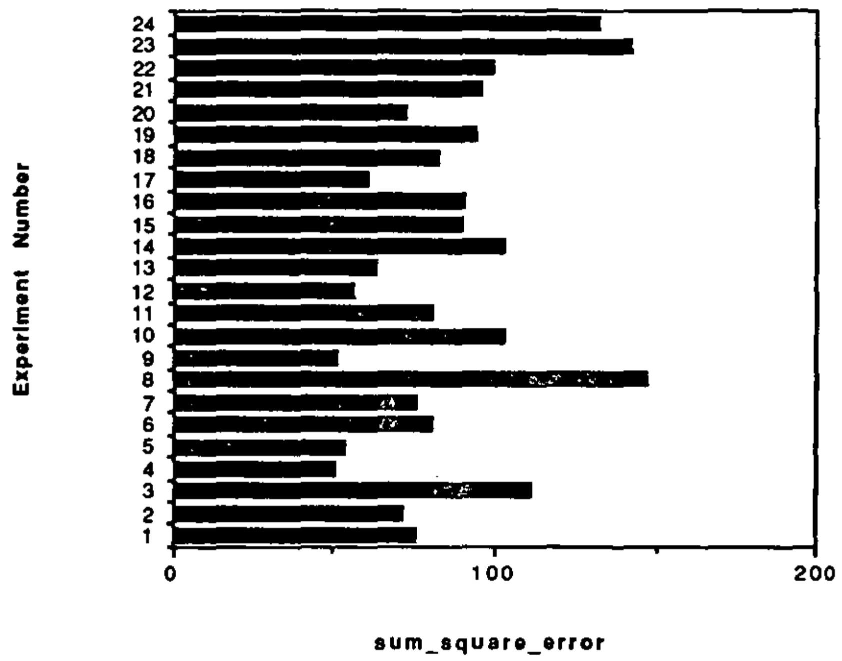
Graph C.2 a. Sum squared error values of four input ghosted neural networks after 1000 iterations



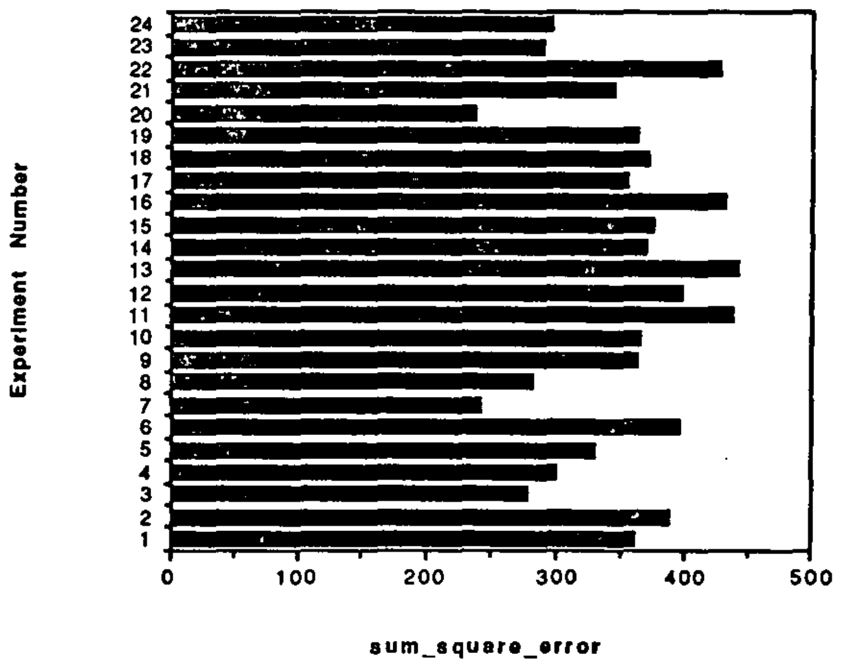
Graph C.2 b. Sum squared error values of five input ghosted neural networks after 1000 iterations



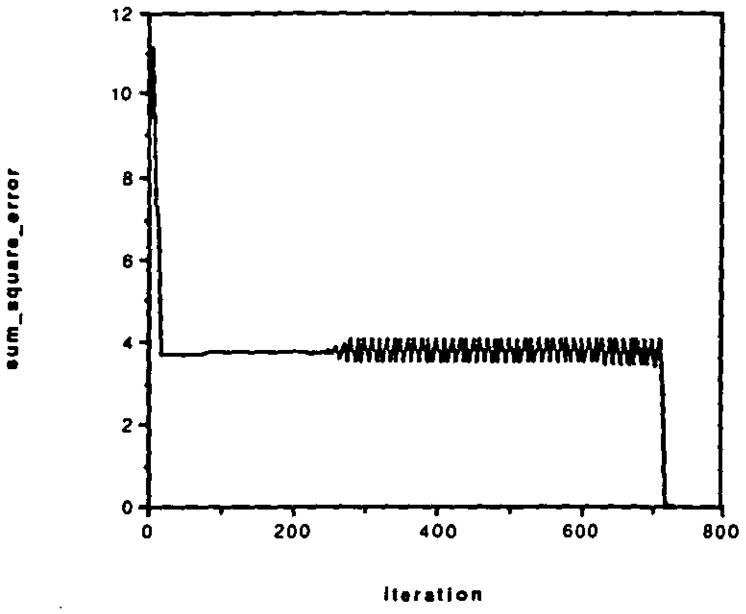
Graph C.2 c. Sum squared error values of six input ghosted neural networks after 1000 iterations



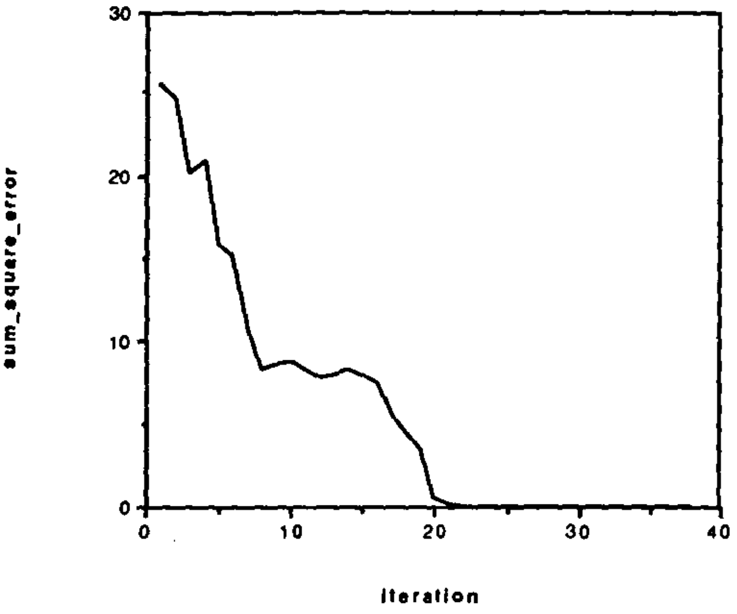
Graph C.2 d. Sum squared error values of seven input ghosted neural networks after 1000 iterations



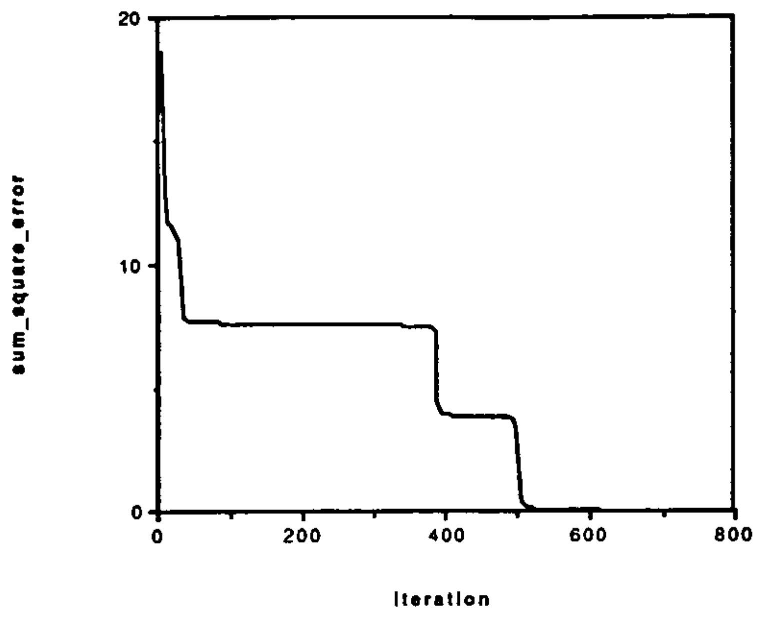
Graph C.2 e. Sum squared error values of eight input ghosted neural networks after 1000 iterations



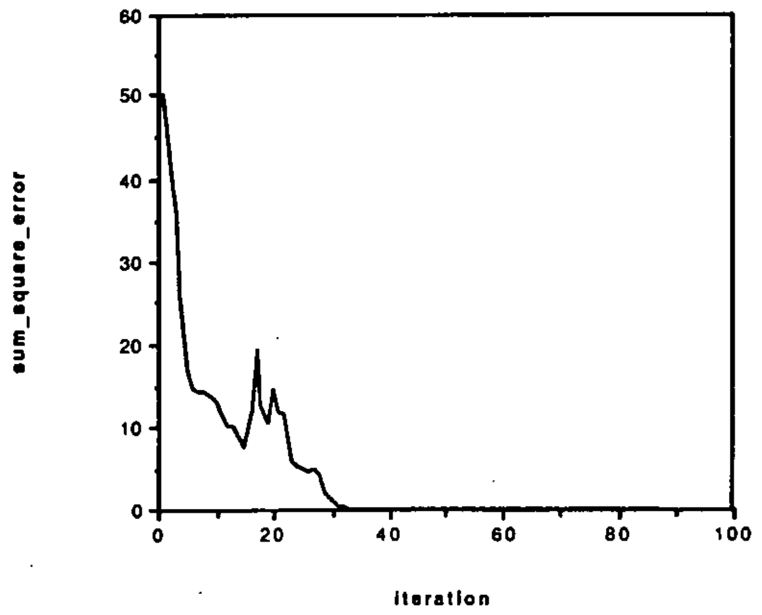
Graph C.3 a. Sum squared error convergence performance of a four input ghosted neural network with a specific training set



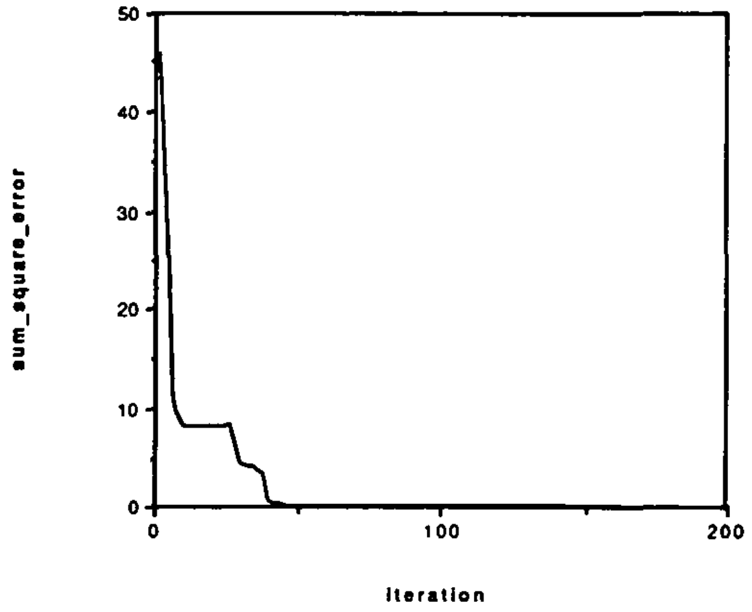
Graph C.3 b. Sum squared error convergence performance of a four input ghosted neural network with a specific training set



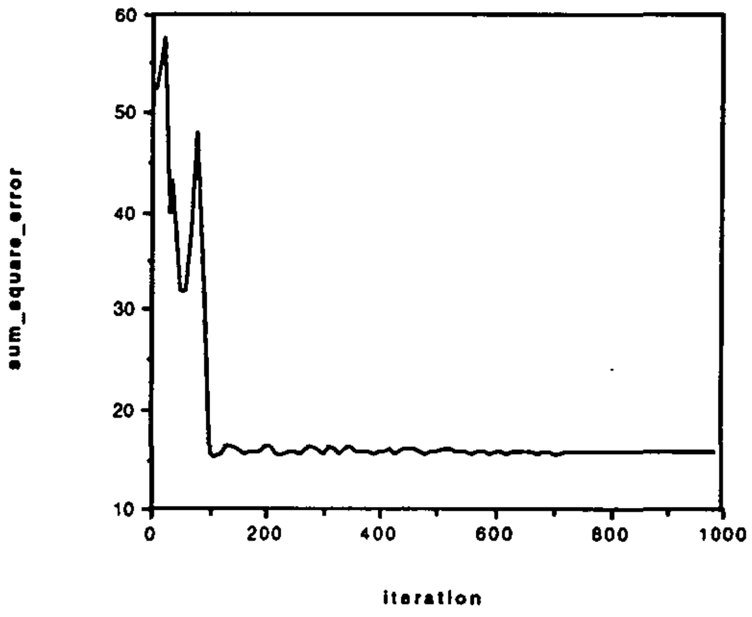
Graph C.3 c. Sum squared error convergence performance of a four input ghosted neural network with a specific training set



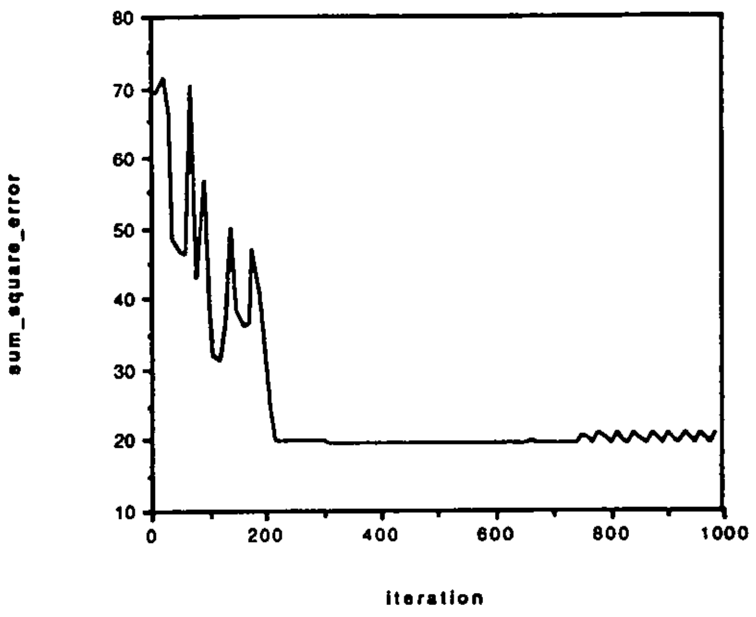
Graph C.3 d. Sum squared error convergence performance of a five input ghosted neural network with a specific training set



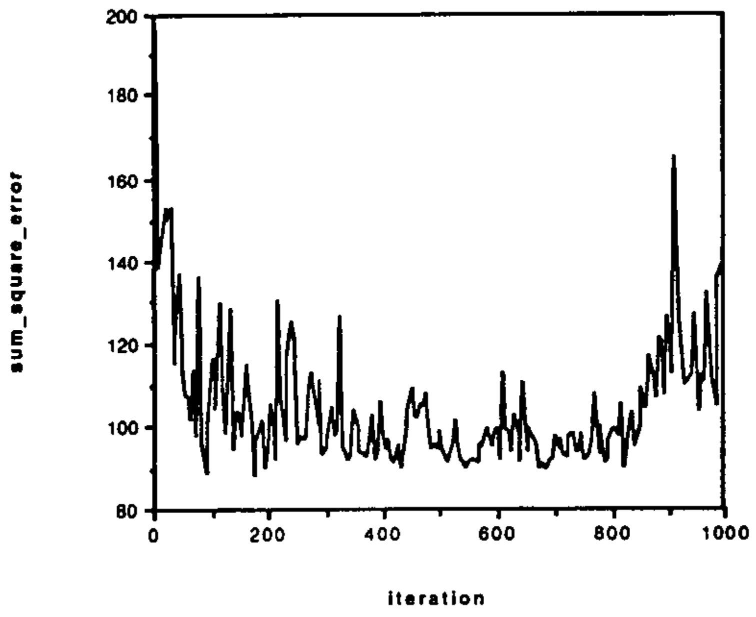
Graph C.3 e. Sum squared error convergence performance of a five input ghosted neural network with a specific training set



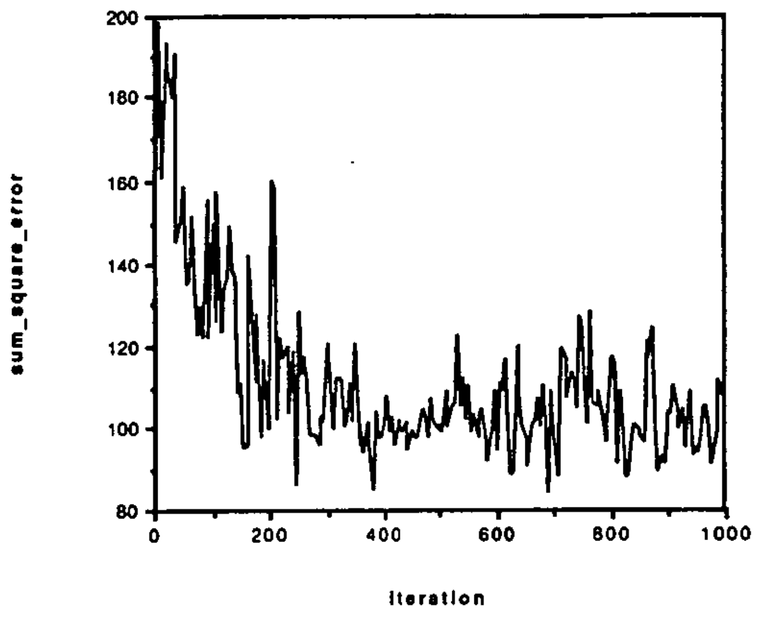
Graph C.3 f. Sum squared error convergence performance of a six input ghosted neural network with a specific training set



Graph C.3 g. Sum squared error convergence performance of a six input ghosted neural network with a specific training set



Graph C.3 h. Sum squared error convergence performance of a seven input ghosted neural network with a specific training set



Graph C.3 i. Sum squared error convergence performance of a seven input ghosted neural network with a specific training set

Appendix D. Encapsulated Sandwich Nets

Aim:

To design an encapsulated sandwich network model of odd parity with three and four inputs. To observe the convergence performance of the odd parity model when trained on even parity data.

Method:

The following encapsulated models of odd parity with three and four inputs were designed. The first layer of nodes (for the three input case) consist of the hidden nodes that isolate the point (1,1,1) giving the output 1, and the point(-1,-1,-1) giving the output -1, and the parallel hyperplane between them. The first layer of nodes (for the four input case) consist of the hidden nodes that isolate the point (1,1,1,1) giving the output -1, and the point (-1,-1,-1,-1) giving the output -1, and the two parallel hyperplanes between them.

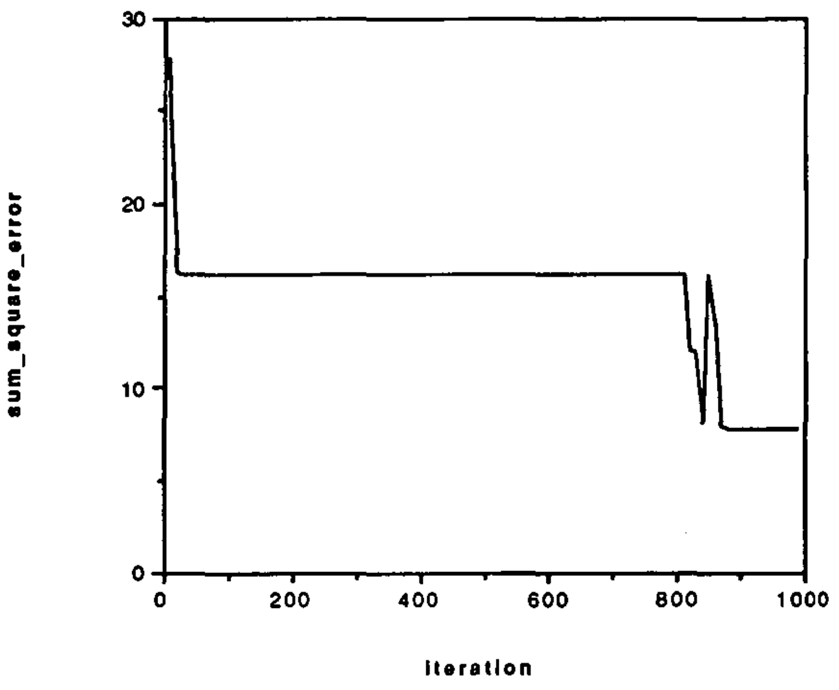
Results:

The encapsulated models of odd parity were seen to be correct. They correctly modelled odd parity over all the training points. The convergence performance of the training algorithm on odd parity neural network models being trained over even parity are shown in graphs D.1 a & b.

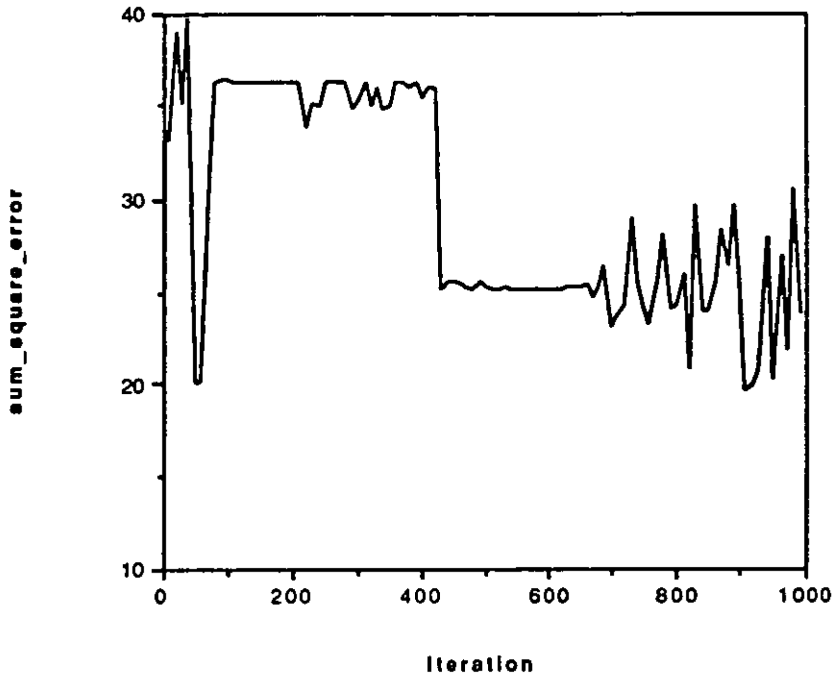
Analysis:

To construct an even model of parity, all the weight values of either the output layer, the sandwich layer or the first hidden layer must be negated. The sum squared error performance compares unfavourably with the experiments of appendix A4. This reflects the extra structure of encapsulated sandwich nodes, which inhibits disruptive training from inconsistent training data.

Graphs:



Graph D.1 a. Sum squared error convergence properties of the encapsulated model of three input odd parity when trained over even parity



Graph D.1 b. Sum squared error convergence properties of the encapsulated model of four input odd parity when trained over even parity

Appendix E1. Amalgamation Schemes for Neural Networks

Schemes for amalgamating lower dimensional schemes of neural networks are presented. Such schemes are useful in determining minimal network topologies and give insights into the construction and design of neural networks.

Schematic representation of nodes

Schematic diagrams of nodes are presented. These allow complex network structures to be succinctly represented.

The horizontal line across the axis represents a splitting plane, making a contribution in each projection. (fig E1.1a)

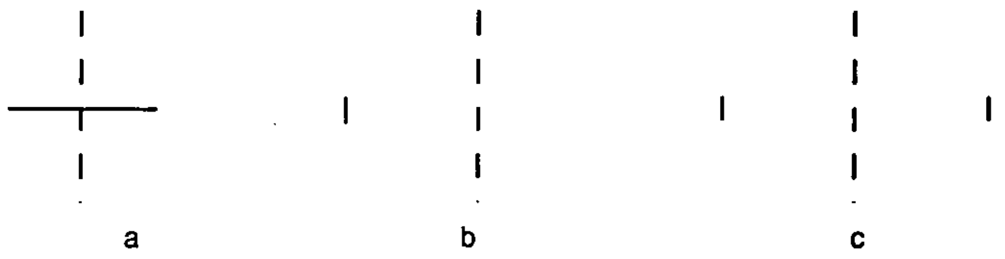


Fig E1.1 a. Schematic diagram of a node split across an axis (the dotted line), that is a node which does not reduce, b. Schematic diagram of a node split which reduces across an axis, c. Schematic diagram of a two reducing nodes split across an axis

Fig E1.1b shows a single small line which represents a reducing plane that only contributes to one half of the split. Fig E1.1c shows a schematic diagram of two reducing planes contributing in opposite sides of the split.

Square nets

Square nets are single output neural networks with the same number of hidden nodes as input nodes. A k-representation of a k input transformation consists of k input nodes and k hidden nodes and so forms a square net.

Lemma

If we have a k node representation of a k input node transformation, the tight representations of the k-1 dimensional splits will consist of k-2 splitting nodes and one reducing node each.

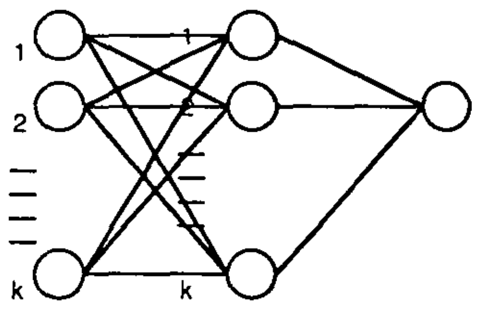


Fig E1.2 A square neural net in which the number of hidden nodes equals the number of input nodes

Proof

Given a k dimensional space represented by k nodes (see fig E1.2), we choose an axis and split the problem. The k nodes are then projected into the k-1 dimensional spaces, viewing them all as splitting nodes. Assuming the k-1 dimensional problem can be solved in k-1 nodes that is the k-1 dimensional representation is tight, only k-1 nodes need contribute in the projection (see fig E1.3). Of the original k nodes one node does not contribute in each split, and so there are two reducing node. The other k-2 nodes are either splitting nodes or reducing nodes.

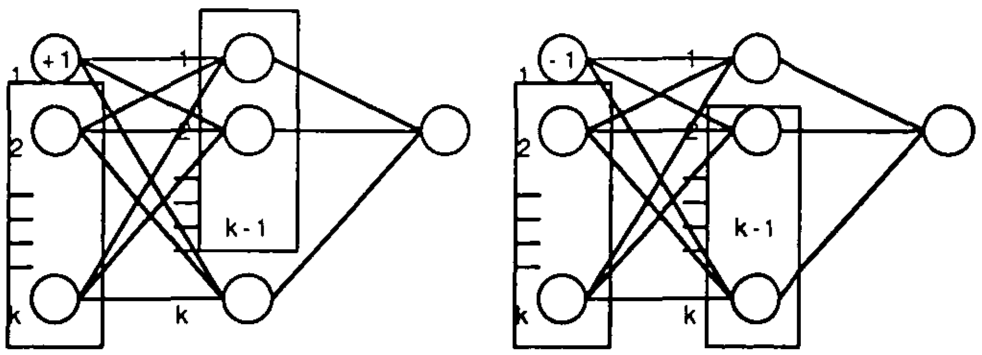


Fig E1.3 Nodes that contribute in a split across a single axis

The two splits share at least k-2 nodes (see fig E1.4). This is the case since if they share less, then the requirement that the k-1 dimensional problem is tight is violated. If they share k-1 nodes then the k dimensional problem is not full and only requires k-1 nodes to represent the problem.

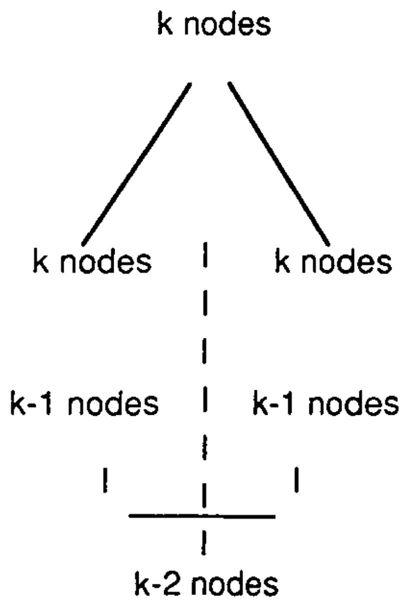


Fig E1.4. Schematic diagram of the number of nodes that split or reduce across an axis. k-2 nodes are shared by both splits, that is they are splitting nodes, while the two remaining nodes only contribute to one of the splits, they are reducing nodes

Amalgamation of consistent nodes

There is no difference between amalgamating reducing nodes or splitting nodes. A k dimensional node s-consistent with those k-1 dimensional nodes we are amalgamating must be constructed. Given either a reducing or splitting node it can be viewed as being defined over the whole k dimensional space, even though it may in fact only really contribute in a discriminating manner in a subspace (fig E1.5a). Fig E1.5b shows nodes that contribute as both splitting and reducing nodes across the two splits.

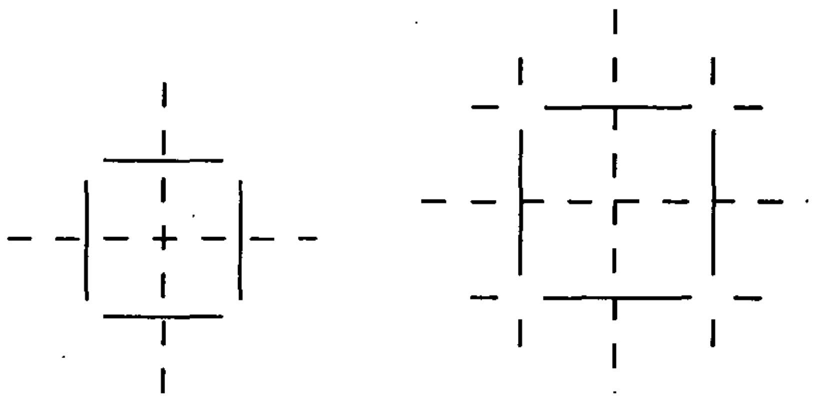


Fig E1.5 a. Schematic diagram showing just nodes splitting across each axis **A** and **B**, b. Schematic diagram showing both splitting and reducing nodes

The nodes of fig E1.5b can be amalgamated if they are consistent over the subspaces in which they contribute in a discriminatory manner. A node that has appeared in the **A** axis split amalgamates with a consistent node from the **B** axis split. Three situations can occur. Two splitting nodes amalgamate, two reducing nodes amalgamate or one splitting node amalgamates a with a reducing node.

Fig E1.6c shows two splitting nodes that have amalgamated, these are called doubly splitting nodes. Fig E1.6a shows two reducing nodes that have amalgamated. These are called doubly reducing nodes. Fig E1.6b shows a reducing node amalgamating with a splitting node. These are called singly splitting or singly reducing nodes.

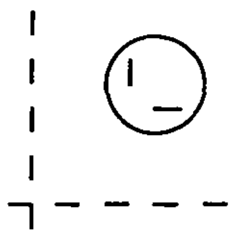


Fig E1.6 a. An amalgamation scheme for two reducing nodes

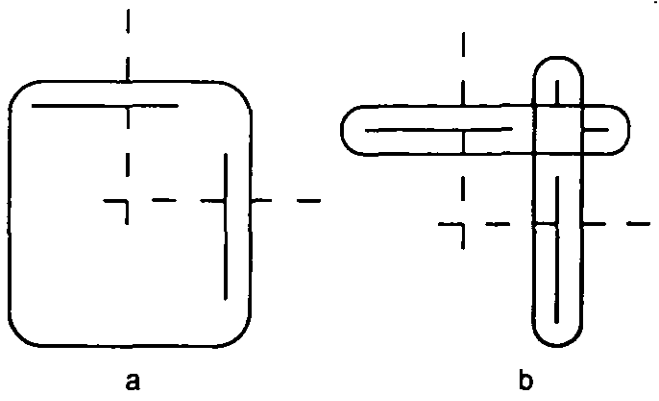


Fig E1.6 b. An amalgamation scheme for a splitting node and a reducing node, c. An amalgamation scheme, with two splitting nodes

By amalgamating the nodes in all four quadrants we are constructing a single node in the full k dimensional problem. By the analysis the k-1 dimensional nodes are proved to be m-consistent with another node in their complement space. The amalgamation of the nodes in the four quadrants result in four types of amalgamated nodes in the k dimensional space.

The simplest form is the doubly reducing node (fig E1.7a). This is a node in the k dimensional space that only contributes in a k-2 dimensional subspace of the problem. The singly splitting node shown in fig E1.7b, is the next case. This is a splitting node (of say the **A** split) that amalgamates with two reducing nodes (of the **B** split). A singly splitting node is a node in the k dimensional space that only contributes in a k-1 subspace of the problem. The third case occurs when two splitting nodes amalgamate in a quadrant and amalgamate with two reducing nodes in two other quadrants. This case is shown in fig E1.7c and is called a partially reducing or a partially splitting node. A partially splitting node is a node in the k dimensional space that contributes in three k-2 subspaces of the problem. The final case is the doubly splitting node, shown in fig E1.7d. This node is formed by amalgamating four splitting nodes. The doubly splitting node is a node in the k dimensional space that contributes in the whole space.

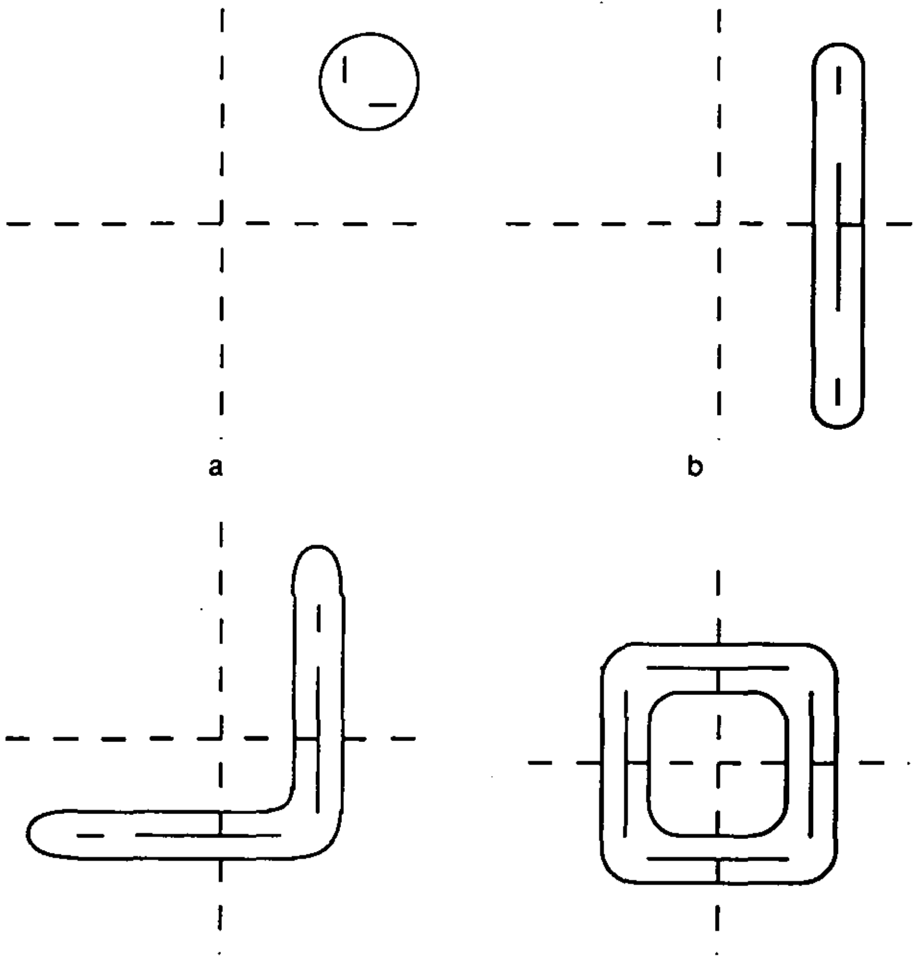


Fig 1.7 a. A doubly reducing node, b. A singly splitting node, c. A partially splitting node, d. A doubly splitting node.

Appendix E2. Amalgamation Scheme for the Case n= 4

We Prove the hypothesis that there exists a tight representation of the worst 4 dimensional transformation using 4 nodes whose 3 dimensional splits are tight requiring 3 nodes.

We can construct full representations of the 4 dimensional transformation by amalgamating two tight 3 dimensional models of the problem. The minimal such model is selected.

We can choose an axis along which to split the problem. Fig E2.1 a & b show two such splits.

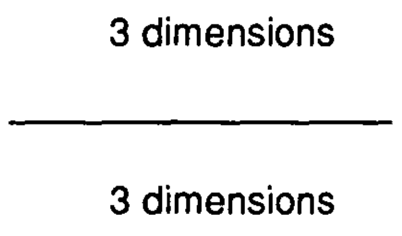
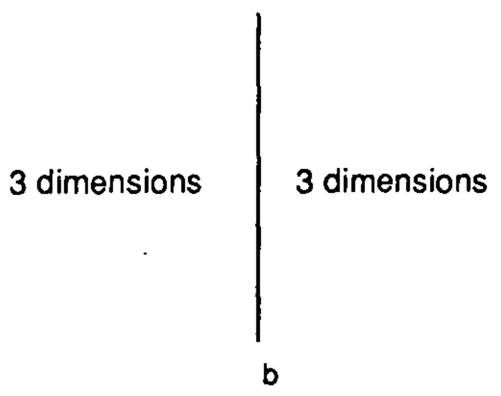


Fig E2.1 a. A split along axis **A**



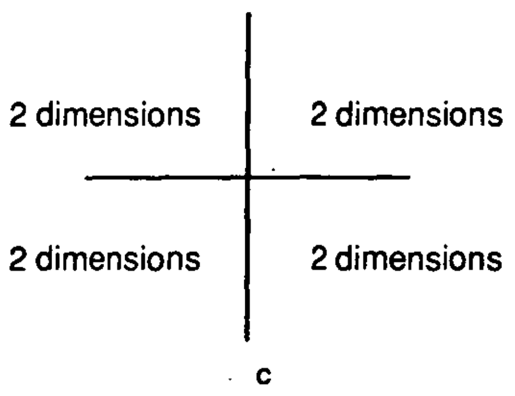
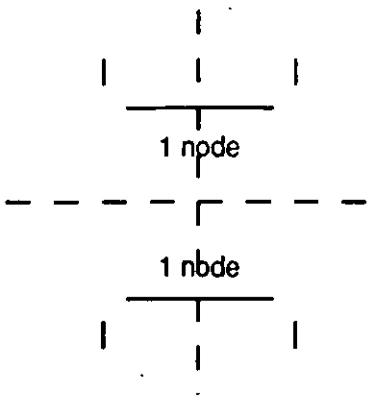
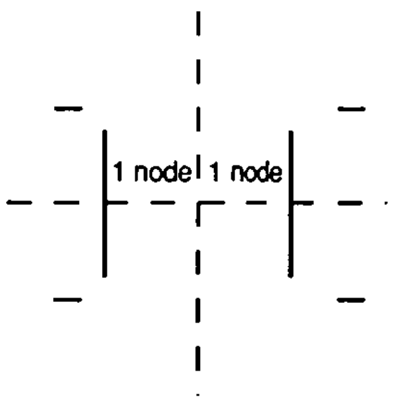


Fig E2.1 b. A split along axis **B**, c. A splits along axes **A** and **B**

Each of these 3 dimensional spaces can be further split into 2 dimensional subspaces by splitting along the other axis. This results in four distinct 2 dimensional spaces from the four 3 dimensional spaces. See fig E2.1c for the schematic representation.

From the analysis of appendix E1, we see that given 3-representations of the 3 dimensional subspaces, the tight 2 dimensional subspaces must consist of 1 splitting node plus one reducing node.





a b

Fig E2.2 a. Schematic diagram of the nodes splitting across axis **B**, b. Schematic diagram of the nodes splitting across axis **A**

Given the minimal model of the worst 4 dimensional problem, say a I-representation, then splitting the problem along axes **A** and **B** would have the nodes appearing in the manner shown in fig E2.2 a & b. Putting all the nodes, in all the projections on one diagram we have the case as shown in fig E2.2c.

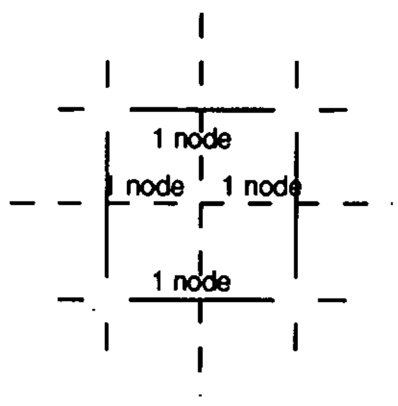


Fig E2.2c. Schematic diagram with both the nodes that split across axis **A** and axis **B**

In each quadrant we have two representations of each node appearing from the two different ways we split the problem so we must produce a scheme to amalgamate the representations. We can amalgamate the nodes in each quadrant since each node in one of the two dimensional splits (say the **A** then **B** split) is consistent with another node from the other two dimensional split (the **B** then **A** split). This is the case since each node in the original I-representation appears at least twice (due to the two different splitting schemes).

The following amalgamation schemes are possible;

For the case in fig E2.3a,

1 splitting node + 4 doubly reducing nodes = 5 nodes.

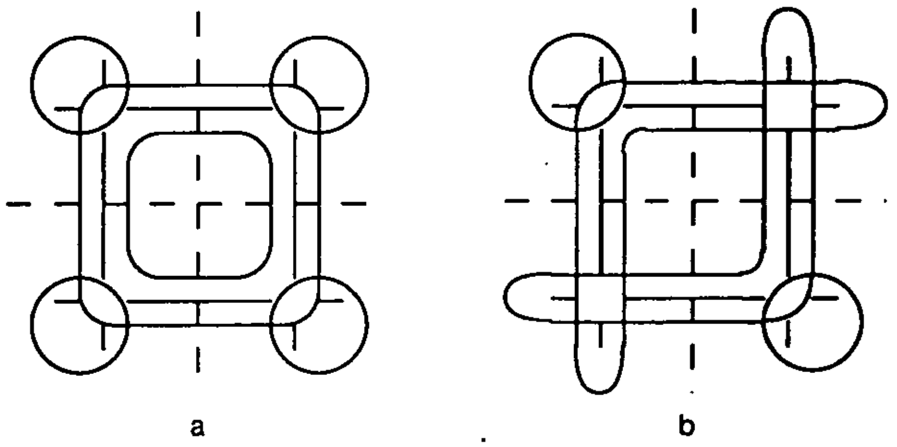


Fig E2.3 a. All the reducing nodes amalgamate and all the splitting nodes amalgamate, b. Two reducing nodes amalgamate, the others amalgamate to form partially splitting nodes

For the case in fig E2.3b,

2 partial splitting nodes +2 doubly reducing nodes = 4 nodes.

For the case in fig E2.3c,

1 partial splitting nodes + 1 doubly reducing nodes +2 singly splitting node = 4 nodes.

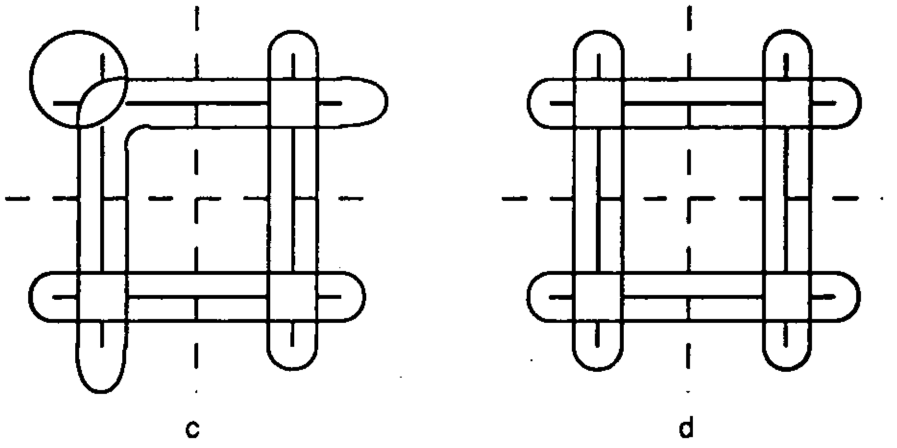


Fig E2.3 c. One reducing node amalgamates, one partially splitting one is formed, and two singly splitting nodes are formed, d. All the nodes formed split an axis singly

For the case in fig E2.3d,

4 singly splitting node = 4 nodes.

Of all the possibilities, we have only one amalgamation scheme that violates the hypothesis (fig E2.3a). This case is considered further. (see fig D16)

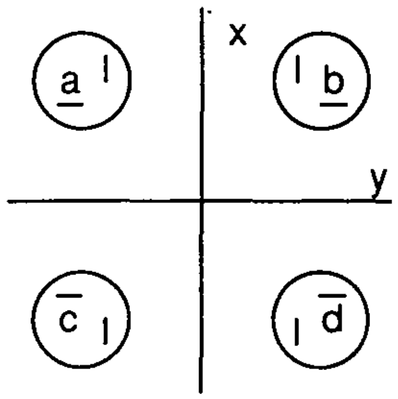


Fig E2.4 a. Unique labelling of the reducing nodes

Considering the splitting of the problem along the axes x and y, we can label the nodes formed as shown, ignoring the splitting ones (fig E2.4a). Doing the same along the axes y and z gives us the situation as shown below (fig E2.4b). If this were not the case then the splits with respect to the axes z and y would form one of the other amalgamation schemes, so proving the hypothesis. The important point to note is that the reducing nodes a, b, c and d can not be one of the splitting planes in this new projection since these splitting planes do not reduce with respect to the y axis, which the nodes a, b, c and d do.

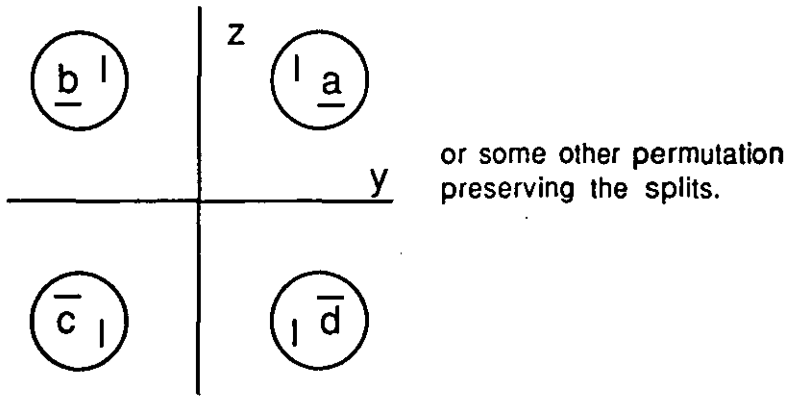


Fig E2.4 b. Permutation of the nodes, preserving the unique labelling

We continue the procedure over the remaining axis, which gives us the result that the nodes a, b, c and d reduce in all the projections, that is they select just one point of the 4 dimensional hypercube. In this case a different amalgamation scheme is possible.

Three of the nodes can be selected and isolated from the rest of the points using two planes, forming a sandwich. One of the nodes is untouched, but the other three can be represented by these two new nodes. Therefore one of the nodes in the original formulation is redundant, so proving the result. For this case n= 4, four points can be isolated by just two high dimensional nodes since four points form a three dimensional hyperplane in a four dimensional space. Using these two nodes, the total number of nodes in this representation will be three. If this were the case, the fact that we are modelling the worst possible case (we already know that parity requires 4 nodes), would be violated. Therefore this representation and amalgamation scheme can not occur.

So the hypothesis that there exists a tight representation of the worst 4 input transformation using 4 hidden nodes whose 3 dimensional splits are tight is proved.

Appendix E3. Amalgamation Scheme for the Case n= k

We prove that given the minimal representation of the worst k transformation whose k-1 dimensional and k-2 dimensional representations are tight can not have more than k hidden nodes. This is proved by examining the possible amalgamation scheme in the tight representations of the k-2 dimensional splits.

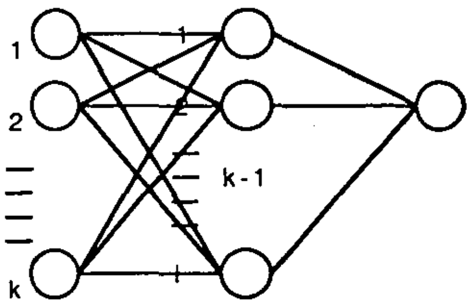


Fig E3.1 General neural net representation of a Boolean transformation

Given a k dimensional problem, with its minimal representation, the I-representation (fig E3.1), we can choose two distinct axes in which to split the problem into lower dimensional problems. (fig E3.2 a & b)

k-1

---

k-1

Fig E3.2 a. A split along axis **A**

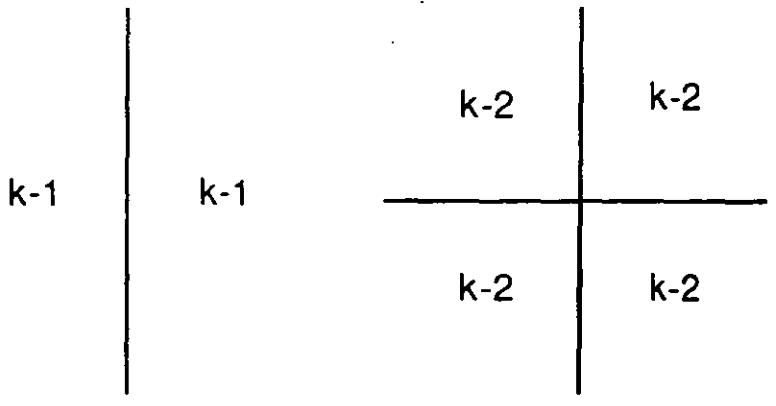


Fig E3.2 b. A split along axis **B**, c. A split along axes **A** and **B**

Each of these k-1 dimensional spaces can be further split into k-2 dimensional subspaces by splitting along the other axis. This results in four distinct k-2 dimensional spaces from the four k-1 dimensional spaces. See fig E3.2c for the schematic representation and fig E3.3 a- d, for the network representation.

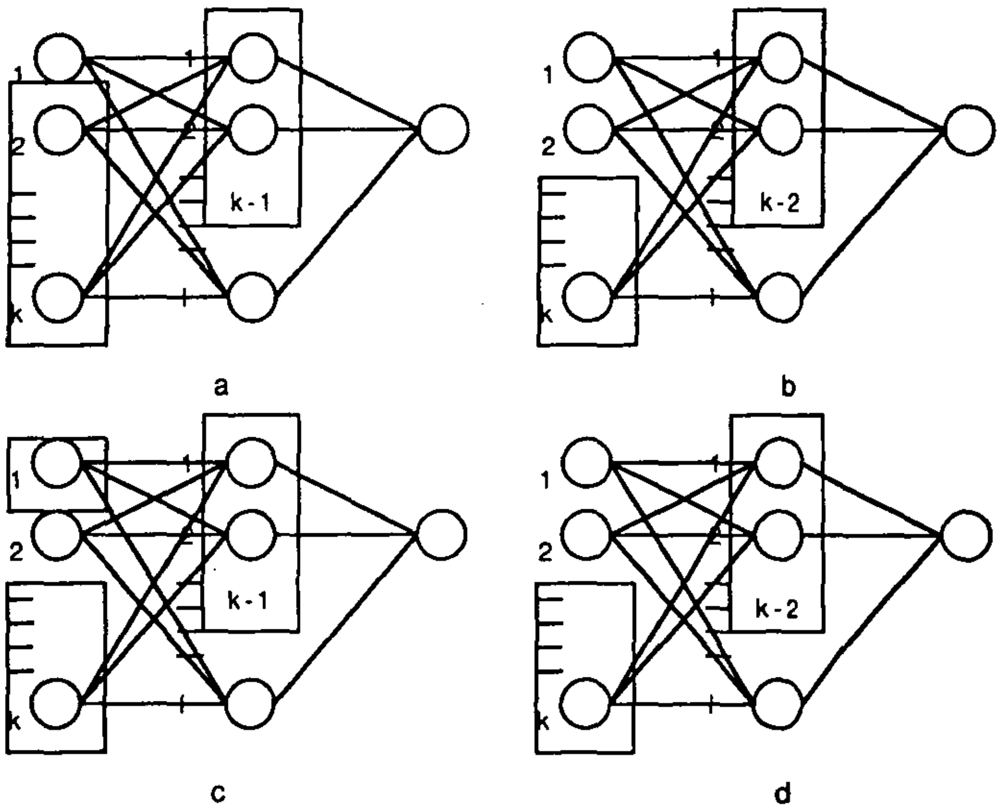
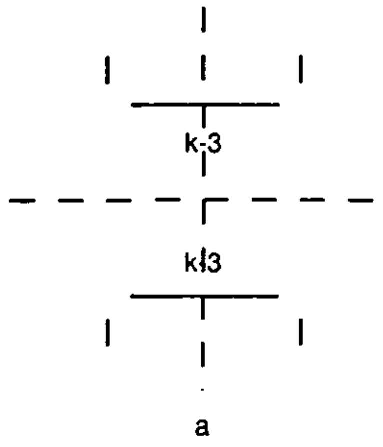


Fig E3.3 Network representation of a split along two axes a. **A**, and c. **B**. The two different orders of applying the splits, b. **A** then **B**, and d. **B** then **A**

From the analysis of appendix E1, we see that given k-1 representations of the k-1 dimensional subspaces, the tight k-2 dimensional subspaces must consist of k-3 splitting nodes plus one reducing node. This is shown by fig E3.4 a & b.



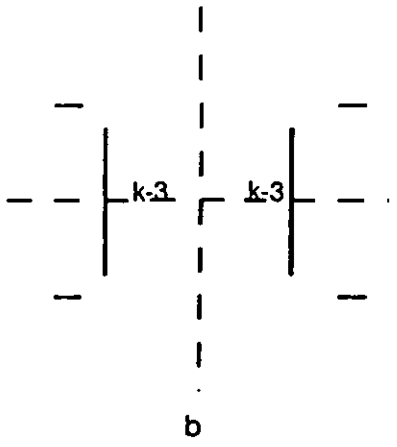


Fig E3.4 a. Schematic diagram of the nodes splitting across axis **B**, b. Schematic diagram of the nodes splitting across axis **A**

If we had a model of the k dimensional problem, say a l-representation, then splitting the problem along the two axes **A** and **B** would have the nodes appearing in the manner in fig E3.4 a & b. Putting all the nodes, in all the projections on one diagram we have the nodes as shown in fig E3.4 c.

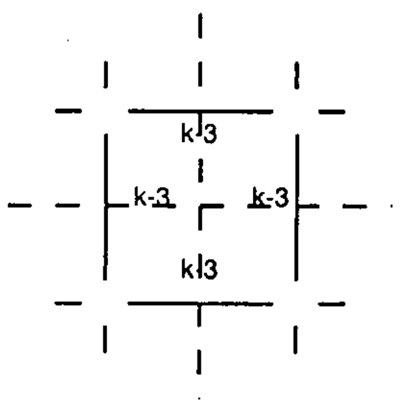


Fig E3.4 c. Schematic diagram with both the nodes that split across axis **A** and axis **B**

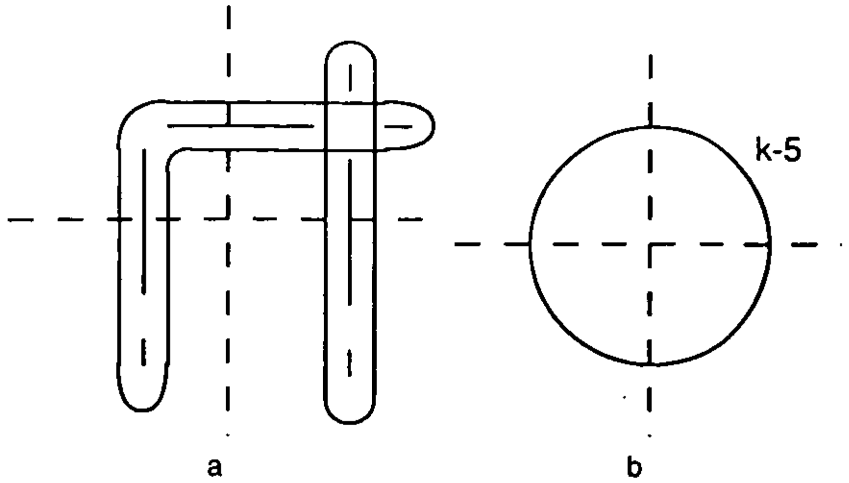


Fig E3.5 a. Singly splitting and partially reducing nodes, these do not form doubly splitting nodes, b. The k-5 doubly splitting nodes that are formed when splitting nodes amalgamate together

In each quadrant we have at least two representations of each node appearing from the two different ways we split the problem so we must produce a scheme to amalgamate the representations.

We can amalgamate the various nodes if they are consistent over the relevant subspaces. The consistency of the nodes are satisfied by the fact that the subspaces are tight. Each k-1 dimensional subproblem is (k-1)-representable, while each k-2 dimensional subproblem is (k-2)-representable and so perpendicular splits of two k-1 representations in a k-2 subspace must be consistent. That is each node in each representation must have a corresponding consistent node in the other and so can be amalgamated.

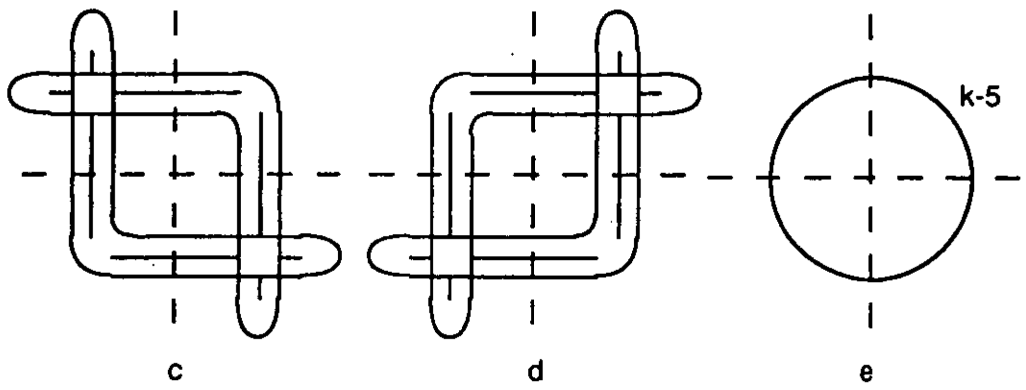


Fig E3.5 c. Two partially reducing nodes, d. Two other partially reducing nodes, e. k-5 doubly splitting nodes

The amalgamation scheme in which the least number of doubly splitting nodes appear is when a splitting node in one k-1 dimensional space is in fact a reducing one in its complement, these nodes are partially reducing nodes or singly splitting nodes (see fig E3.5a). This means that since each k-1 space has only two reducing nodes, a total of at least k-5 nodes are still in fact splitting nodes across the two reducing axes. (fig E3.5b)

For the example in fig E3.5 c- e, we have ;
k-5 splitting nodes + 4 partially reducing nodes = k-1 nodes. This does not violate the induction hypothesis. The case were we have three partially reducing nodes is similar.

If there are less than three partially splitting nodes in the amalgamated representations then there will be at least k-4 doubly splitting nodes. The following amalgamation schemes are possible
For the case in fig E3.6a,
k-4 original splitting nodes + 1 splitting node + 4 doubly reducing nodes = k+1 nodes.

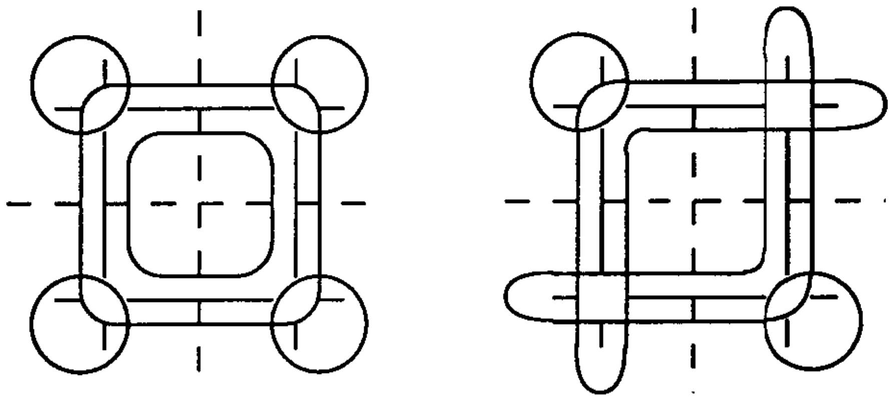


Fig E3.6 a. All the reducing nodes amalgamate and all the splitting nodes amalgamate, b. Two reducing nodes amalgamate, the others amalgamate to form partially splitting nodes

For the case in fig E3.6b,
k-4 original splitting nodes +2 partial splitting nodes +2 doubly reducing nodes = k nodes.
For the case in fig E3.6c,

k-4 original splitting nodes +1 partial splitting nodes + 1 doubly reducing nodes +2 singly splitting node = k nodes.

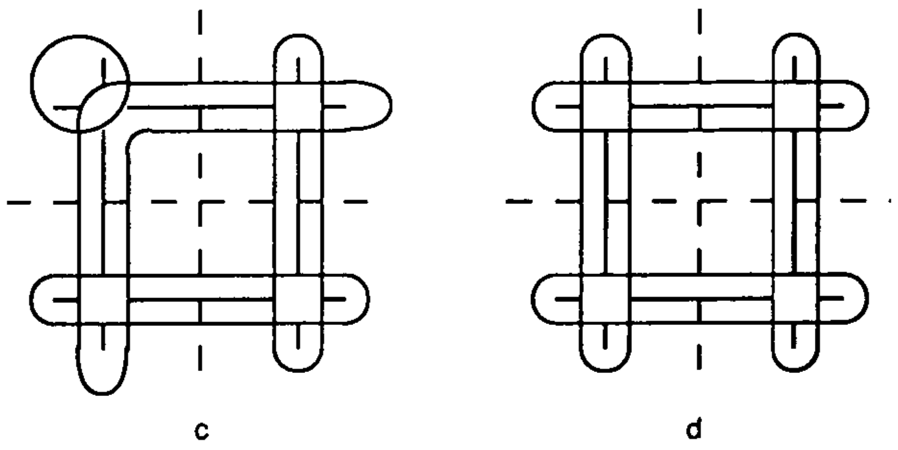


Fig E3.6 c. One reducing node amalgamates, one partially splitting one is formed, and two singly splitting nodes are formed, d. All the nodes formed split an axis singly

For the case in fig E3.6d,

k-4 original splitting nodes +4 singly splitting node = k nodes.

Of all the possibilities, we have only one amalgamation scheme that violates the hypothesis (fig E3.6a). This case can be proved to be non minimal in an identical manner to that of appendix E2. So the hypothesis that the minimal representation of the worst k transformation whose k-1 dimensional and k-2 dimensional representations are tight can not have more than k hidden nodes is proved.

Appendix F1. Quantisation Experiments

Aim:

To identify the ability of a single layer of nodes to model the arbitrary quantisation of real valued input to a Boolean output.

Method:

The data sets where generated by a seeded pseudorandom scheme. The data consisted of a real valued input value followed by the Boolean output values. The real valued output is given by the formula;

Output= (Seed - 505)/ 505,

Nextseed= (Seed * 101 * 992) mod 1009,

and the Boolean outputs were generated in the order (1,1,...,1,1) then (1,1,...,1,-1) and (1,1,...,-1,1) and (1,1,...,-1,-1) and iteratively to (-1,-1,...,-1,-1).

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one input and n quantisation nodes are defined by 2*n weight values.

The experiments were generated with a pseudorandom scheme. Twenty four pseudorandom neural network configurations were selected to be trained on pseudorandom data sets given by the formula;

Dataseed= 7 * Inputnodes,

Netseed= 11 * Inputnodes,

Experiment is net(Netseed) trained on data(Dataseed),

Nextdataseed= (Dataseed * 101 * 992) mod 1009,

Nextnetseed= (Netseed * 101 * 992) mod 1009.

The temperature value of 0.1 and a learning rate of 0.01 was used. The experiments were run for 1000 iterations. The sum squared error convergence characteristics are shown in graphs F1.1 a- c and table F1.1.

Results:

The sum squared error values of the arbitrary input quantisation neural network increases as the number of input quantisation nodes increases. Most of the experiments do not converge completely within the 1000 iterations examined. Table F1.1 shows the mean and variance of the sum squared error performance for the specific number of quantisation nodes used. Graphs F1.1 a- c show the sum squared error performance for each of the experiments. These values will be used to compare the performance of different neural network structures at modelling arbitrary real valued transformations.

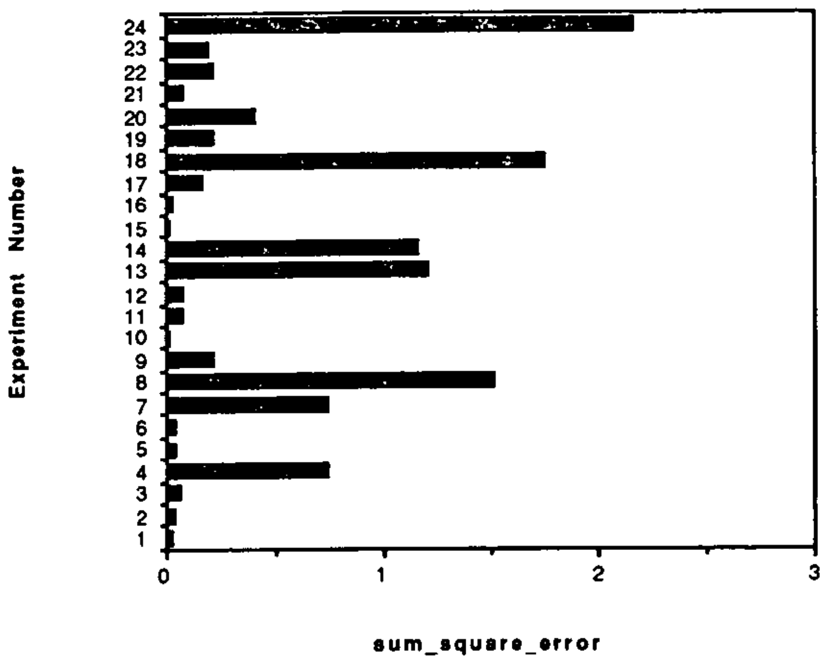
Experiment	Mean	Variance
2 output case	0.465	0.594
3 output case	1.722	1.444
4 output case	7.010	3.239

Table F1.1. Mean and variance of the sum squared error convergence after 1000 iterations for the single layer quantisation neural network structure

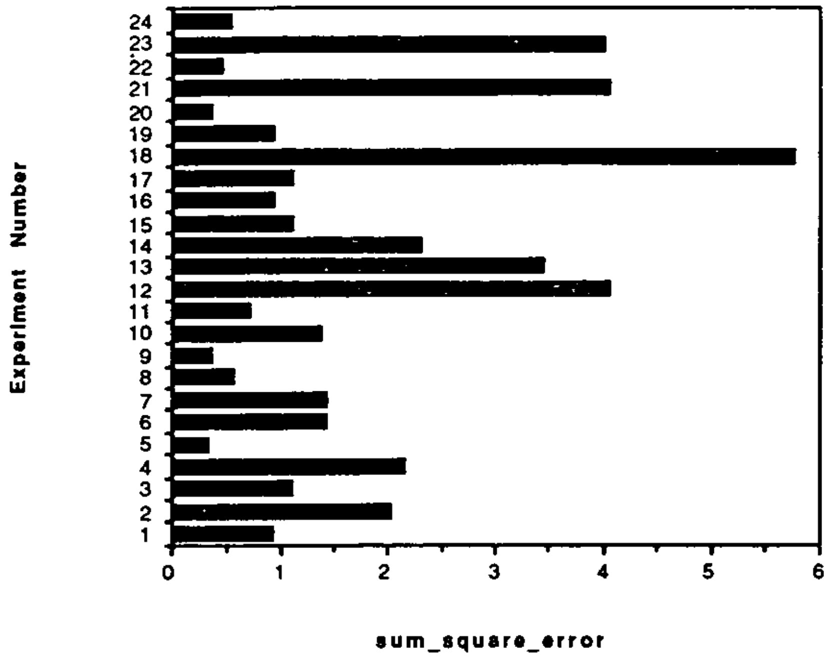
Analysis:

The results show that as the number of quantisation nodes are increased the single layer quantisation neural network's ability to model the arbitrary real input quantisation decreases. A single layer quantisation neural network structure can only model linear quantisation transformations. Since the training data are random input quantisations many will be non linear. The poor convergence performance of the single layered neural network quantisation structure can be explained by its inability to model the nonlinear transformations. The deterioration in performance as the number of quantisation nodes increase are explained by the increased probability of nonlinear transformations for the larger number of quantisation nodes.

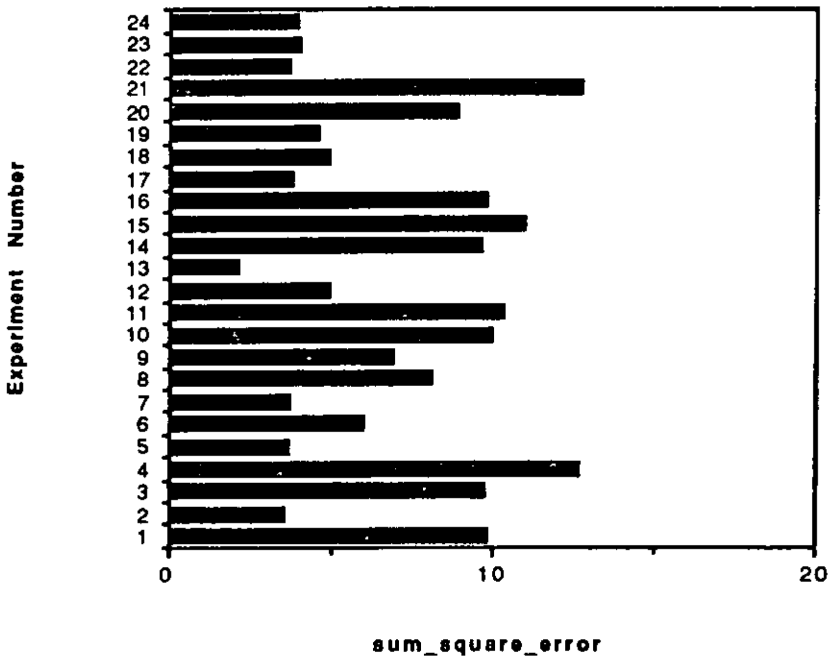
Graphs:



Graph F1.1 a. Sum square error performance of the single layer quantisation neural network structure with two quantisation nodes after 1000 iterations



Graph F1.1 b. Sum square error performance of the single layer quantisation neural network structure with three quantisation nodes after 1000 iterations



Graph F1.1 c. Sum square error performance of the single layer quantisation neural network structure with four quantisation nodes after 1000 iterations

Appendix F2. Decoding Experiments

Aim:

To identify the ability of a single nodes to model the arbitrary decoding of Boolean input to a real valued output.

Method:

The data sets were generated by a seeded pseudorandom scheme. The data consisted of the Boolean input values generated as the outputs of the experiments in appendix F1 followed by a real valued output generated as the inputs of the experiments in appendix F1.

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one decoding output node and n input nodes are defined by n +1 weight values.

The experiments were generated with a pseudorandom scheme. Twenty four random neural network configurations were selected to be trained on random data sets in an identical manner to that of appendix F1. The temperature value of 0.1 and a learning rate of 0.01 was used. The experiments were run for 1000 iterations. The sum squared error convergence characteristics are shown in graphs F2.1 a- c and table F2.1.

Results:

The single layer decoding neural network structure's sum squared error values increases as the number of input nodes increases. The performance is significantly worse than that of the single layer quantisation neural network. Table F2.1 shows the mean and variance of the sum squared error performance for the specific number of input nodes used. Graphs F2.1 a- c show the sum squared error performance for each of the experiments.

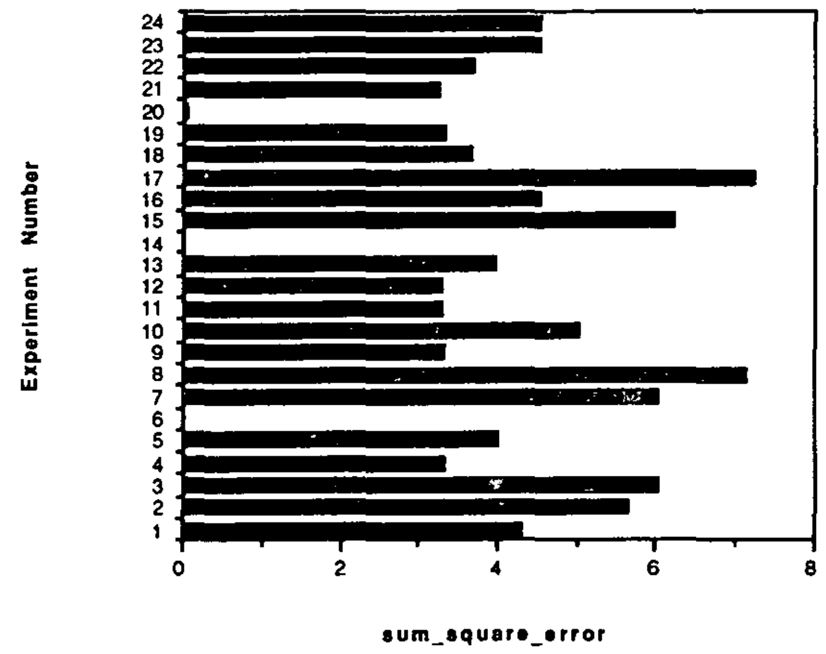
Experiment	Mean	Variance
2 input case	4.019	1.917
3 input case	13.325	3.085
4 input case	35.814	2.748

Table F2.1 Mean and variance of the sum squared error after 1000 iterations for the single layer decoding neural network structure

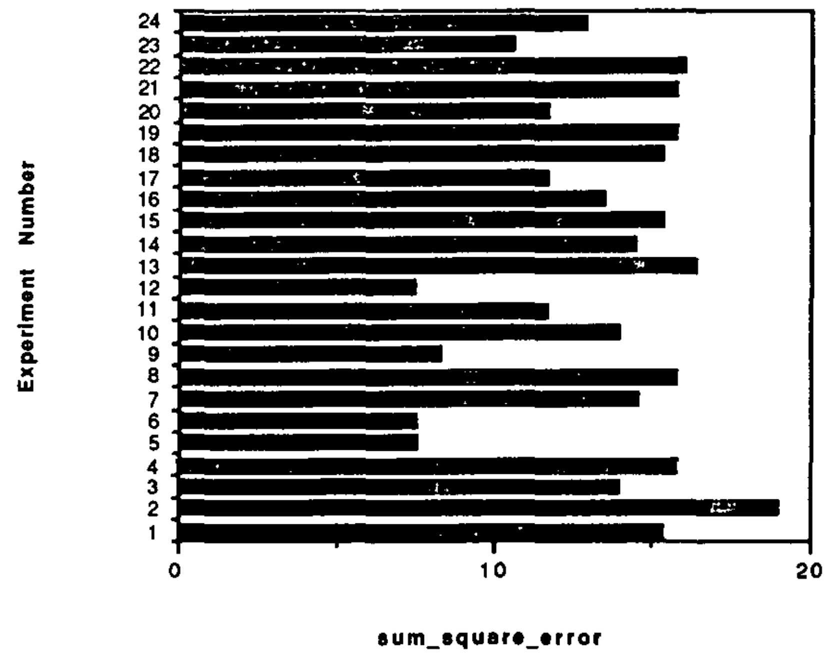
Analysis:

A single layer decoding neural network cannot model arbitrary transformations from a Boolean to a real valued space unless the transformation is monotonic over the Boolean space. That is the Boolean space does not possess any exclusive **o r** properties. The sum squared error performance of the decoding system is worse than that of the quantisation system since the condition that input quantisations are linear transformations are more probable than the condition that the output decoding is monotonic over the input space.

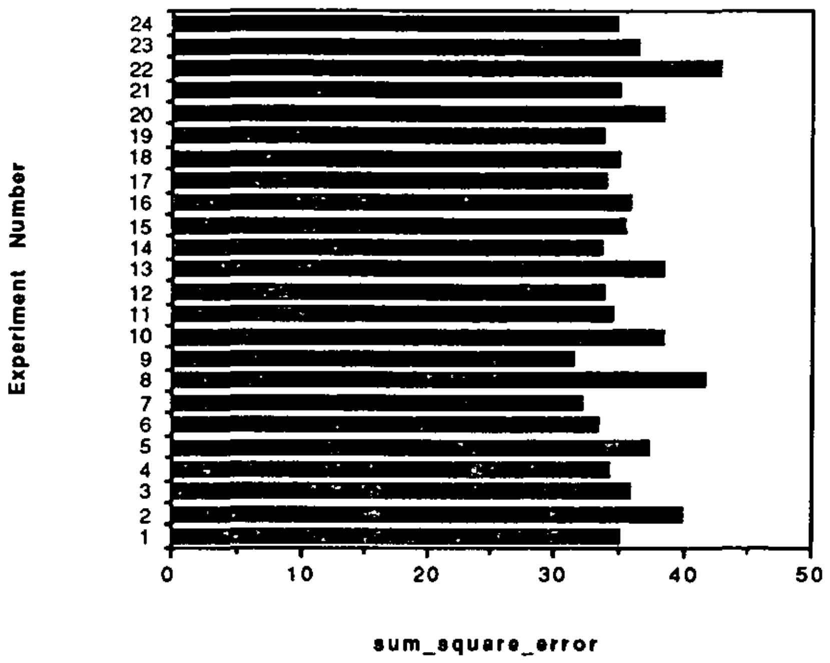
Graphs:



Graph F2.1 a. Sum square error performance of the single layer decoding neural network structure with two decoding nodes after 1000 iterations



Graph F2.1 b. Sum square error performance of the single layer decoding neural network structure with three decoding nodes after 1000 iterations



Graph F2.1 c. Sum square error performance of the single layer decoding neural network structure with four decoding nodes after 1000 iterations

Appendix F3. Multilayer Quantisation Experiments

Aim:

To identify the ability of three layers of nodes to model the arbitrary quantisation of real valued input to a Boolean output.

Method:

The data sets were identical to those in appendix F1.

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one input and three layers of n quantisation nodes are defined by 2*n*(n+1) weight values.

The experiments were generated with a pseudorandom scheme. Twenty four random neural network configurations were selected to be trained on random data sets in an identical manner to that of appendix F1. The temperature value of 0.1 and a leaming rate of 0.01 was used. The experiments were run for 1000 iterations. The iterative convergence characteristics are shown in graphs F3.1 a & b. The sum squared error convergence performance is shown in graphs F3.2 a- e and table F3.1.

Results:

The sum squared error values of the multilayer quantisation neural network increases as the number of nodes increases. The sum squared error values are less than those of the single layer quantisation neural networks.

Two input case;	12/ 24 experiments converged within 1000 iterations.
Three input case;	11/ 24 experiments converged within 1000 iterations.

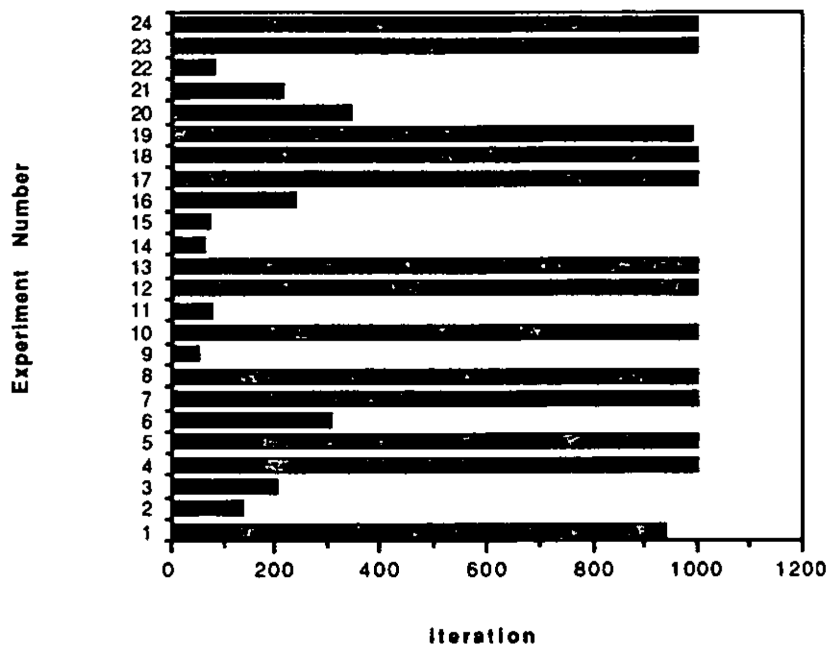
Experiment	Mean	Variance
2 output case	0.290	0.572
3 output case	0.439	0.701
4 output case	1.120	2.044
5 output case	4.211	4.119
6 output case	22.249	10.910

Table F3.1 Mean and variance of the sum squared error after 1000 iterations for the three layer decoding neural network structure

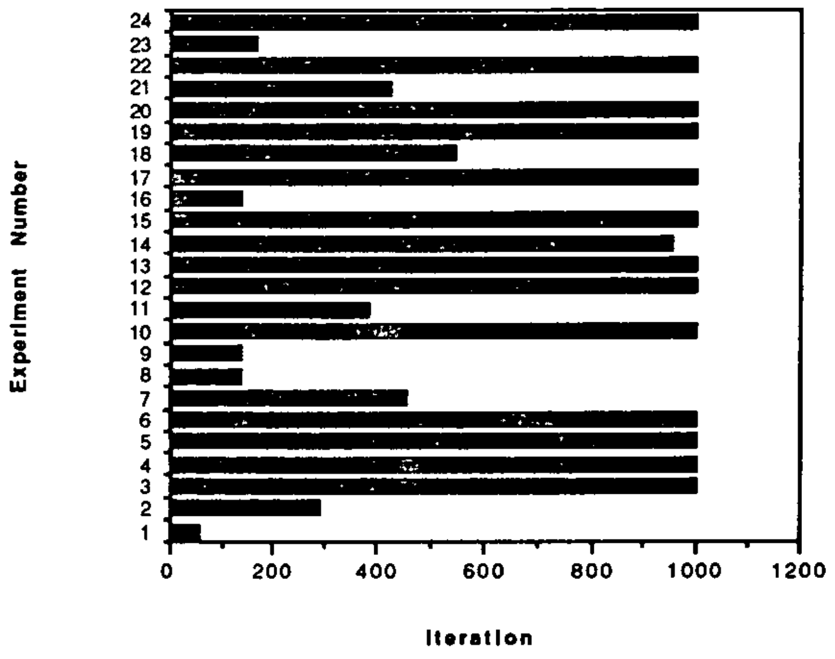
Analysis:

The ability of three layered quantisation neural network structures to model arbitrary quantisations is better than that of single layered structures. The representational power increases with the increased number of layers and is demonstrated by the results. The three layered quantisation neural network structure is not limited to modelling linear quantisation transformations.

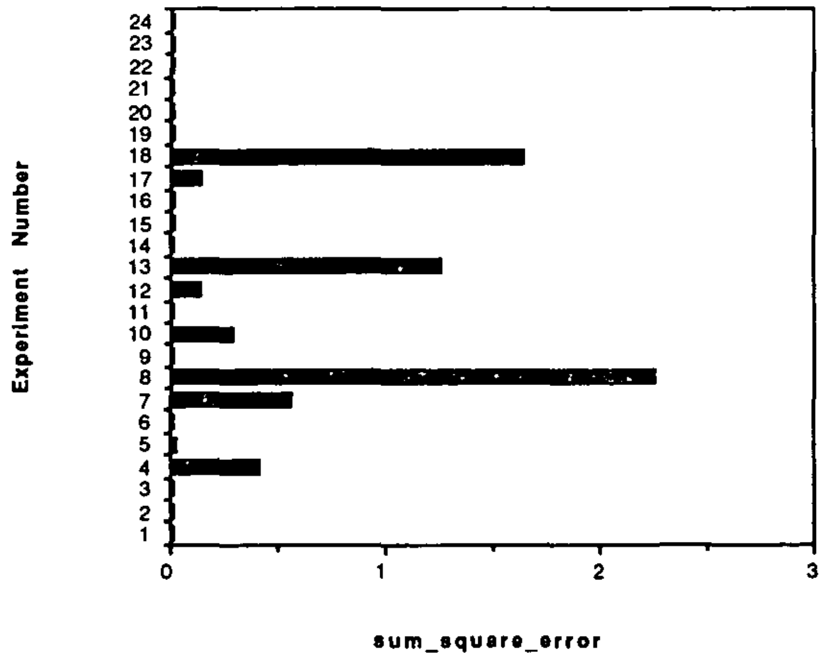
Graphs:



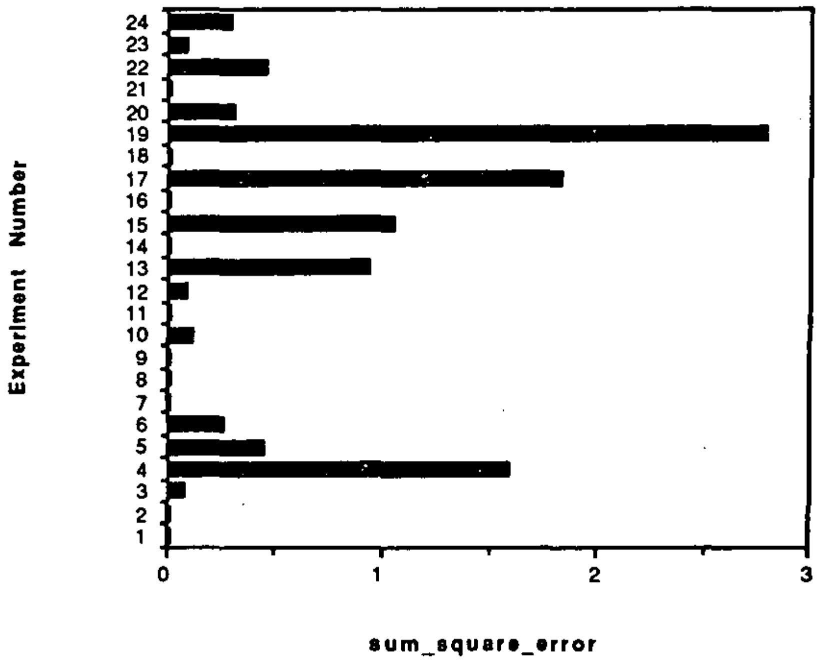
Graph F3.1 a. Number of iterations required for convergence of the three layer quantisation neural network structure with two input and hidden nodes



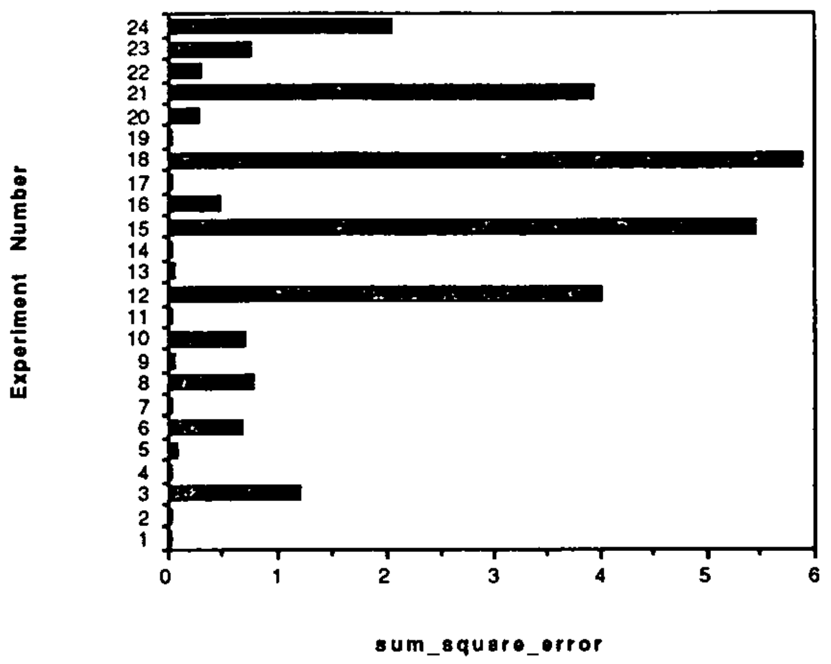
Graph F3.1 b. Number of iterations required for convergence of the three layer quantisation neural network structure with three input and hidden nodes



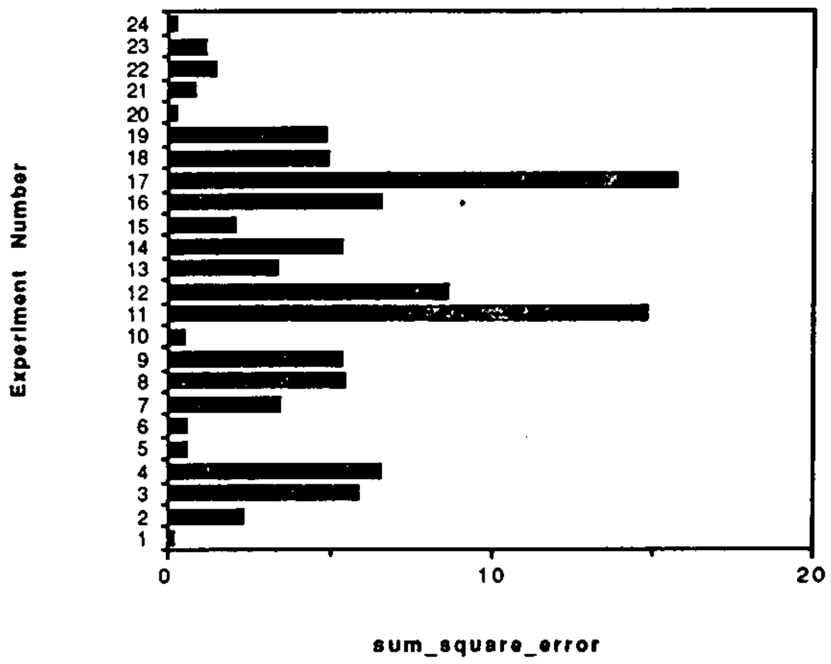
Graph F3.2 a. Sum square error performance of the three layer quantisation neural network structure with two input and hidden nodes after 1000 iterations



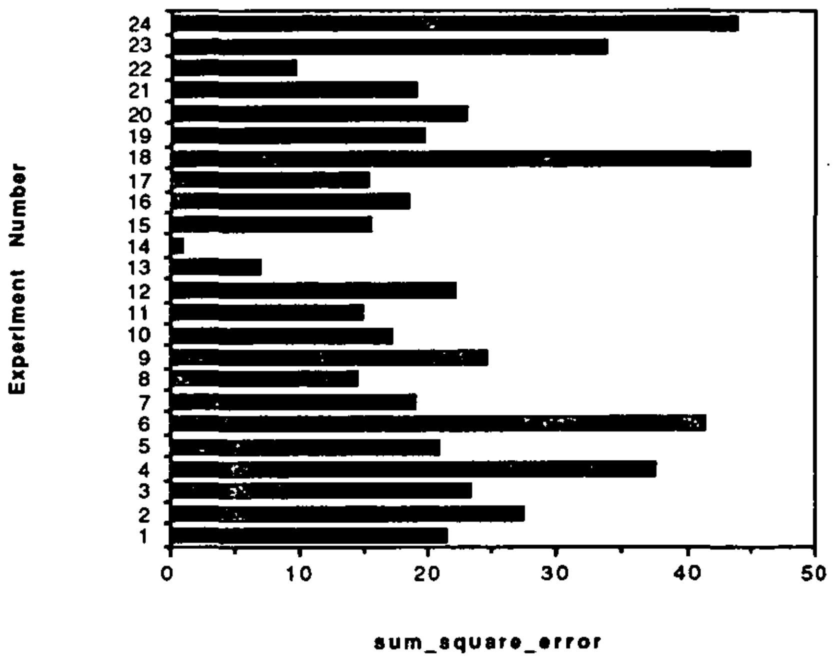
Graph F3.2 b. Sum square error performance of the three layer quantisation neural network structure with three input and hidden nodes after 1000 iterations



Graph F3.2 c. Sum square error performance of the three layer quantisation neural network structure with four input and hidden nodes after 1000 iterations



Graph F3.2 d. Sum square error performance of the three layer quantisation neural network structure with five input and hidden nodes after 1000 iterations



Graph F3.2 e. Sum square error performance of the three layer quantisation neural network structure with six input and hidden nodes after 1000 iterations

Appendix F4. Quantising and Decoding Experiments

Aim:

To identify the ability of three layers of hidden nodes to model the arbitrary quantisation and decoding of a real valued input to a real valued output.

Method:

The data sets were generated by a seeded pseudorandom scheme. The data consisted of a real valued input followed by a real valued output. These were generated via;

Output= (Outputseed - 505)/ 505,

Input= (Inputseed - 505)/ 505,

Nextoutputseed= (Outputseed * 101 * 992) mod 1009,

Nextinputseed= (Intputseed * 101 * 992) mod 1009,

and continued iteratively for all the whole input set, namely 2^n data points.

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one input node three layers of n hidden nodes and one output node are defined by $2*n^2 + 5*n +1$ weight values.

The experiments were generated with a pseudorandom scheme. Twenty four random neural network configurations were selected to be trained on random data sets. The temperature value of 0.1 and a learning rate of 0.01 was used. The experiments were run for 1000 iterations. The convergence characteristics are shown in graphs F4.1 a- d.

Results:

The final sum squared error values after 1000 iterations of the multilayer quantising and decoding neural network increases as the number of hidden nodes increases. The sum squared error values are comparable to those obtained for single layered decoding neural networks in appendix F2.

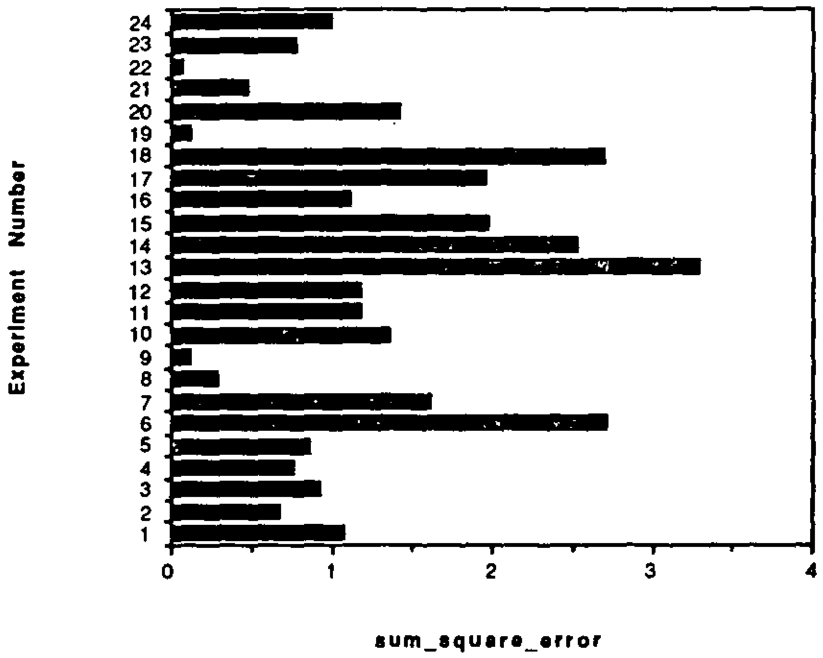
Experiment	Mean	Variance
2 hidden nodes	1.250	0.860
3 hidden nodes	3.409	2.448
4 hidden nodes	7.637	4.738
5 hidden nodes	19.393	8.523

Table F4.1 Mean and variance of the sum squared error after 1000 iterations for the four layer quantisation and decoding neural network structure

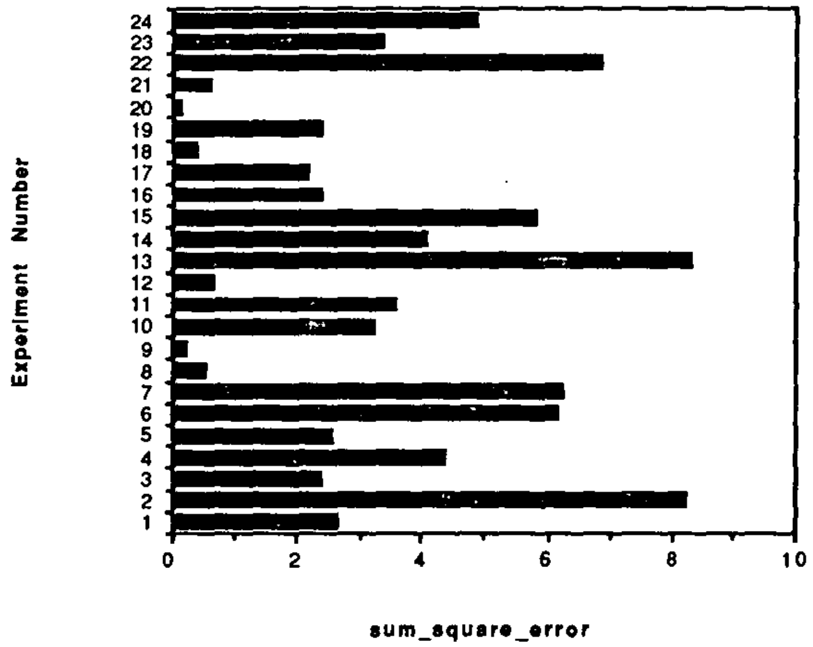
Analysis:

A larger neural network structure has greater representational power than a smaller neural network structure. Given a larger neural network structure, larger training sets can be modelled. Larger training sets provide more conflicting update values and so optimal convergence requires more than the 1000 iterations studied in this experiment. The comparable performance with the single layered decoding neural network reflects the fact that the output layer of the multilayer quantising and decoding neural network is a single layered decoding layer. The sum squared error convergence performance of the multilayered neural network as compared with the single layered decoding neural network will improve with more iterations since the multilayered neural network can manipulate the values fed forward to the decoding layer.

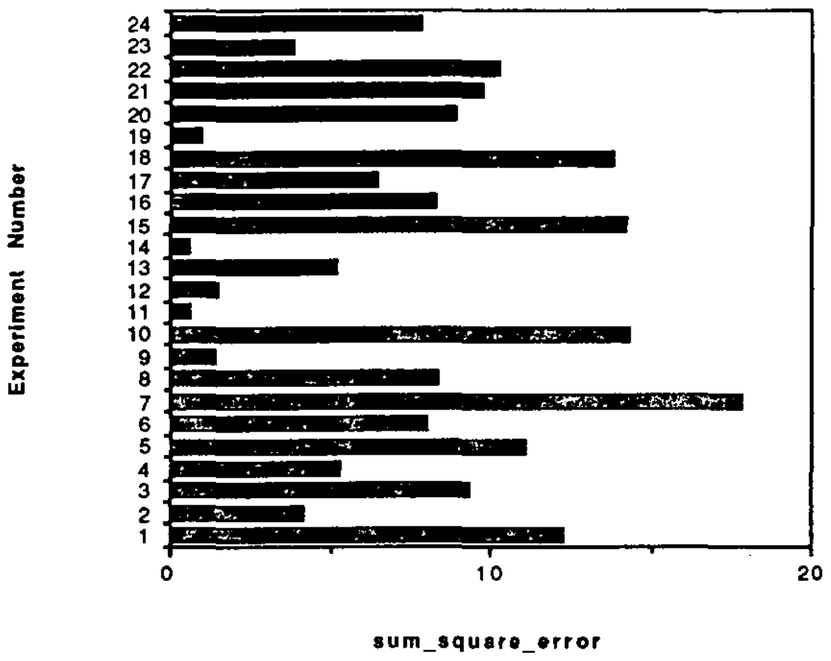
Graphs:



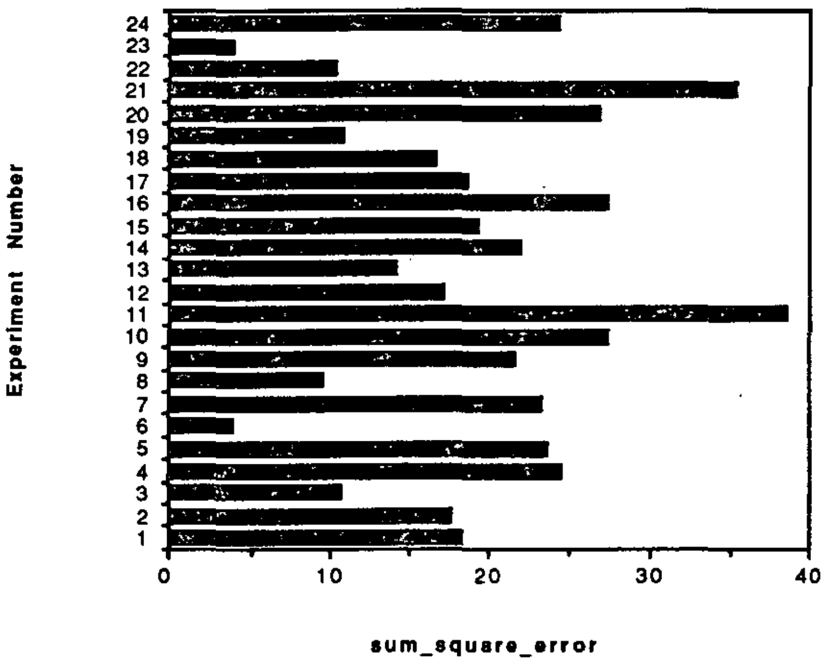
Graph F4.1 a. Sum square error performance of the four layer quantisation and decoding neural network structure with two nodes in each hidden layer after 1000 iterations



Graph F4.1 b. Sum square error performance of the four layer quantisation and decoding neural network structure with three nodes in each hidden layer after 1000 iterations



Graph F4.1 c. Sum square error performance of the four layer quantisation and decoding neural network structure with four nodes in each hidden layer after 1000 iterations



Graph F4.1 d. Sum square error performance of the four layer quantisation and decoding neural network structure with five nodes in each hidden layer after 1000 iterations

Appendix F5. Reduced Input Quantisation Experiments

Aim:

To identify the ability of a single layer of nodes to model the arbitrary quantisation of real valued input to a Boolean output given a reduced training set.

Method:

The data sets were generated by a seeded pseudorandom scheme. The data consisted of a real valued input generated as the outputs of the experiments in appendix F1 followed by the Boolean input values generated as the inputs of the experiments in appendix F1. The reduced training sets that were generated consisted of n*(n+1) points for $n \geq 5$ and 2^n points for $n \leq 4$. This is the same sized training sets for $n \leq 4$ as in appendix F1 and a reduced set for $n \geq 5$.

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one input node and n decoding output nodes are defined by 2*n weight values.

The experiments were generated with a pseudorandom scheme. Twenty four random neural network configurations were selected to be trained on random data sets in an identical manner to that of appendix F1. The temperature value of 0.1 and a learning rate of 0.01 was used. The experiments were run for 1000 iterations. The sum squared error convergence characteristics are shown in graphs F5.1 a- c and table F5.1.

Results:

The final sum squared error values after 1000 iterations of the arbitrary reduced input quantisation neural network increases as the number of input quantisation nodes increases. The sum squared error convergence performance is comparable to those of appendix F1 when the training sets were of identical size, namely $n \leq 4$, while the sum squared error convergence performance is much improved over the reduced training sets $n \geq 5$. Table F5.1 shows the mean and variance of the sum squared error performance for the specific number of input nodes used. Graphs F5.1 a- d show the sum squared error performance for each of the experiments.

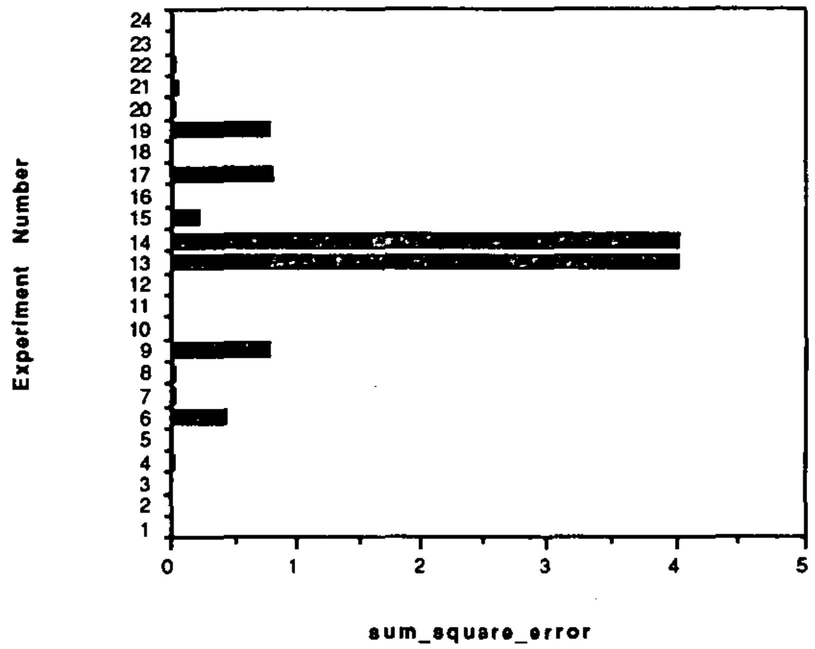
Experiment	Mean	Variance
2 output case	0.471	1.113
3 output case	1.809	1.507
4 output case	7.776	2.669
5 output case	18.067	4.158

Table F5.1 Mean and variance of the sum squared error after 1000 iterations of the reduced input set single layer quantisation neural network structure

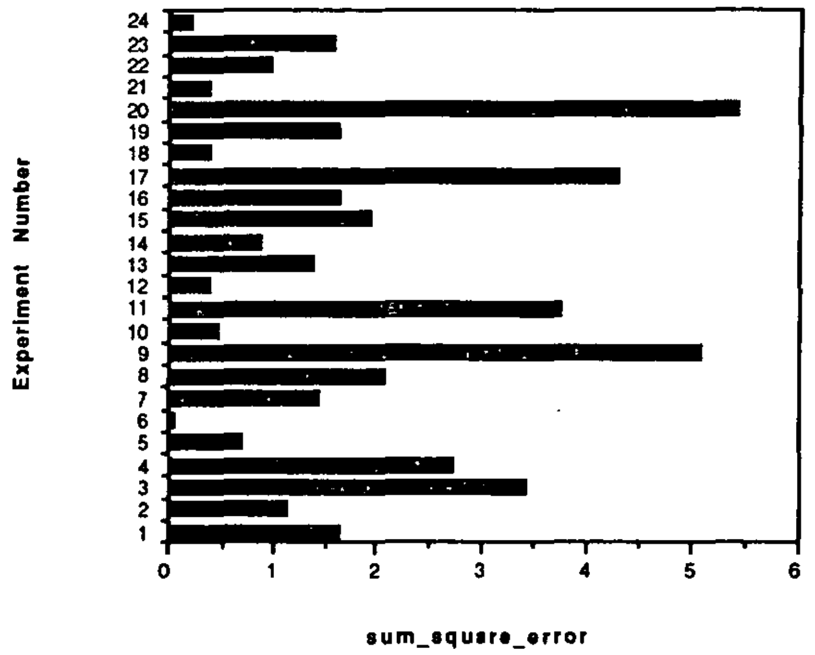
Analysis:

The reduced training sets that were employed in this experiment allow training sets that are linear to be more probable than before. Therefore the single layered neural networks are more likely to converge with the reduced training sets.

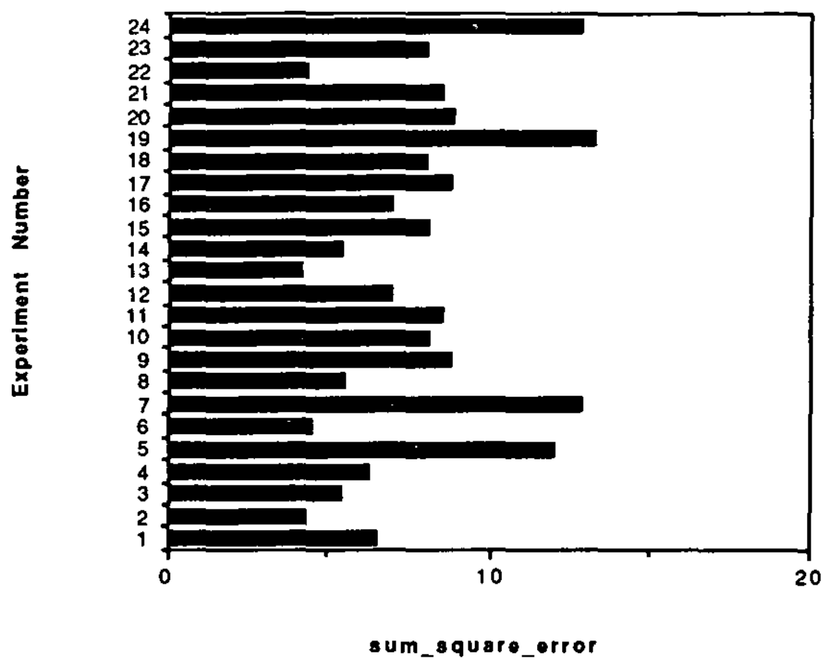
Graphs:



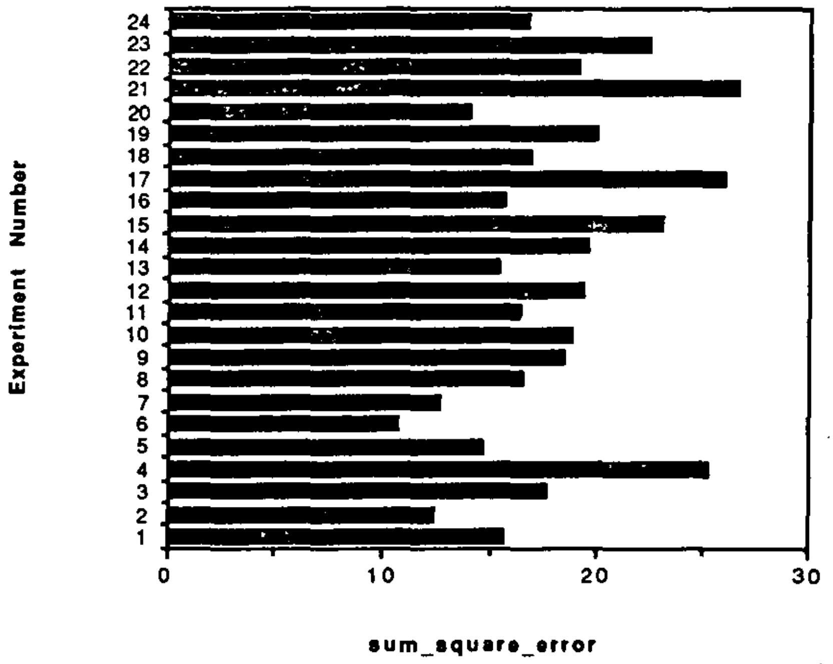
Graph F5.1 a. Sum square error performance of the reduced input set single layer quantisation neural network structure with two quantisation nodes after 1000 iterations



Graph F5.1 b. Sum square error performance of the reduced input set single layer quantisation neural network structure with three quantisation nodes after 1000 iterations



Graph F5.1 c. Sum square error performance of the reduced input set single layer quantisation neural network structure with four quantisation nodes after 1000 iterations



Graph F5.1 d. Sum square error performance of the reduced input set single layer quantisation neural network structure with five quantisation nodes after 1000 iterations

Appendix F6. Reduced Input Decoding Experiments

Aim:

To identify the ability of a single nodes to model the arbitrary decoding of Boolean input to a real valued output over a reduced training set.

Method:

The data sets were generated by a seeded pseudorandom scheme. The data consisted of the Boolean input values generated as the outputs of the experiments in appendix F1 followed by a real valued output generated as the inputs of the experiments in appendix F1. The reduced training sets were defined as in appendix F5.

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one decoding output node and n input nodes are defined by n +1 weight values.

The experiments were generated with a pseudorandom scheme. Twenty four random neural network configurations were selected to be trained on random data sets in an identical manner to that of appendix F1. The temperature value of 0.1 and a learning rate of 0.01 was used. The experiments were run for 1000 iterations. The sum squared error convergence characteristics are shown in graphs F2.1 a- d and table F6.1.

Results:

The sum squared error values of a single layered decoding neural network structure trained on reduced data sets increase as the number of inputs increase. The sum squared error convergence performance is comparable to those of appendix F2 when the training sets were of identical size, namely $n \leq 4$, while the sum squared error convergence performance is much improved over the reduced training sets n≥ 5. Table F6.1 shows the mean and variance of the sum squared error performance for the specific number of quantisation nodes used. Graphs F6.1 a- d show the sum squared error performance for each of the experiments.

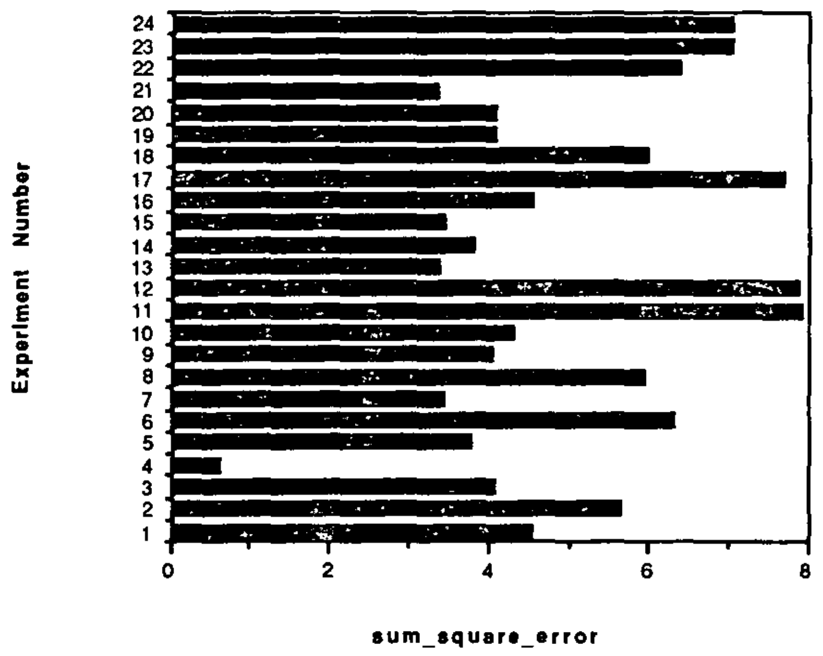
Experiment	Mean	Variance
2 input case	4.952	1.765
3 input case	14.191	1.776
4 input case	35.020	2.647
5 input case	68.590	3.451

Table F6.1 Mean and variance of the sum squared error after 1000 iterations of the reduced input set single layer decoding neural network structure

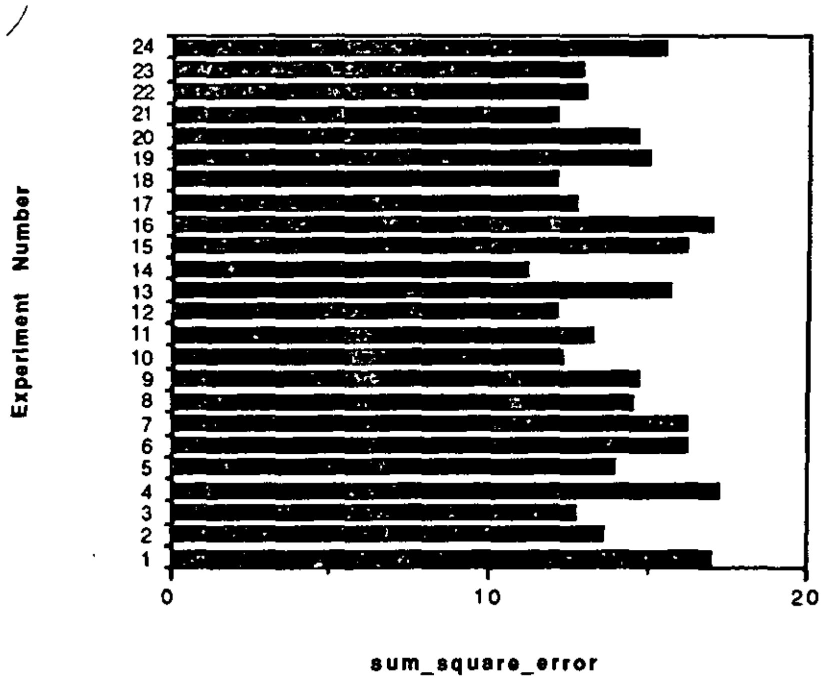
Analysis:

The reduced data sets allow the transformations from Boolean to real valued spaces that are monotonic to be more probable. This means that the training algorithm is more likely to converge given reduced training data sets.

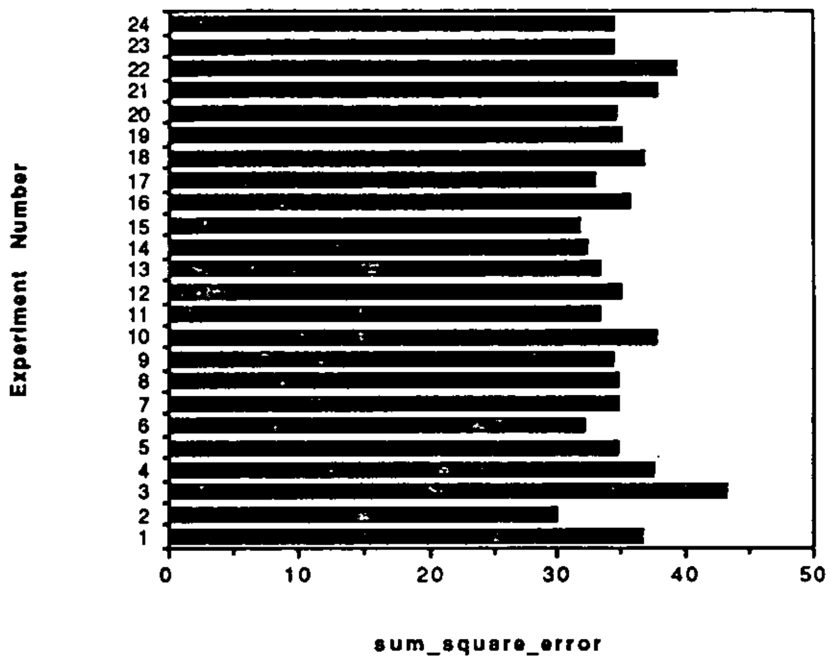
Graphs:



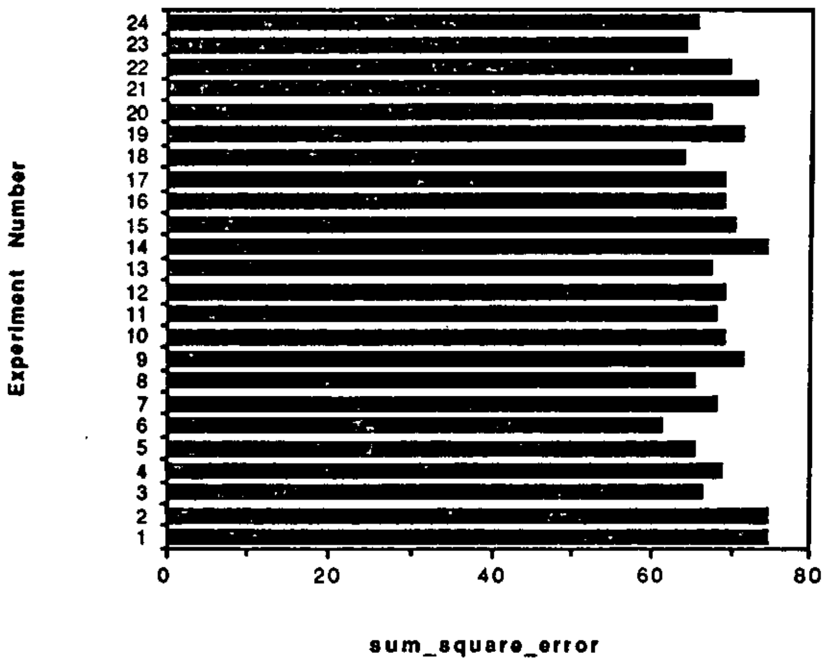
Graph F6.1 a. Sum square error performance of the reduced input set single layer decoding neural network structure with two decoding nodes after 1000 iterations



Graph F6.1 b. Sum square error performance of the reduced input set single layer decoding neural network structure with three decoding nodes after 1000 iterations



Graph F6.1 c. Sum square error performance of the reduced input set single layer decoding neural network structure with four decoding nodes after 1000 iterations



Graph F6.1 d. Sum square error performance of the reduced input set single layer decoding neural network structure with five decoding nodes after 1000 iterations

Appendix F7. Reduced Input Multilayer Quantisation Experiments

Aim:

To identify the ability of three layers of nodes to model the arbitrary quantisation of real valued input to a Boolean output over a reduced training set.

Method:

The data sets were identical to those in appendix F5.

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one input and three layers of n quantisation nodes are defined by $2*n*(n+1)$ weight values.

The experiments were generated with a pseudorandom scheme. Twenty four random neural network configurations were selected to be trained on random data sets in an identical manner to that of appendix F1. The temperature value of 0.1 and a learning rate of 0.01 was used. The experiments were run for 1000 iterations. The iterative convergence characteristics are shown in graphs F7.1 a- c. The sum squared error convergence characteristics are shown in graphs F7.2 a- g and table F7.1.

Results:

The sum squared error values of a multilayered quantisation neural network structure trained on reduced data sets increase as the number of inputs increase. The sum squared error convergence performance is comparable to those of appendix F3 when the training sets were of identical size, namely $n \leq 4$, while the sum squared error convergence performance is much improved over the reduced training sets n≥ 5. Table F7.1 shows the mean and variance of the sum squared error performance for the specific number of output nodes. Graphs F7.2 a- g show the sum squared error performance for each of the experiments.

The sum squared error values of this experiment is much better than those of the single layered quantisation experiment as discussed in appendix F5.

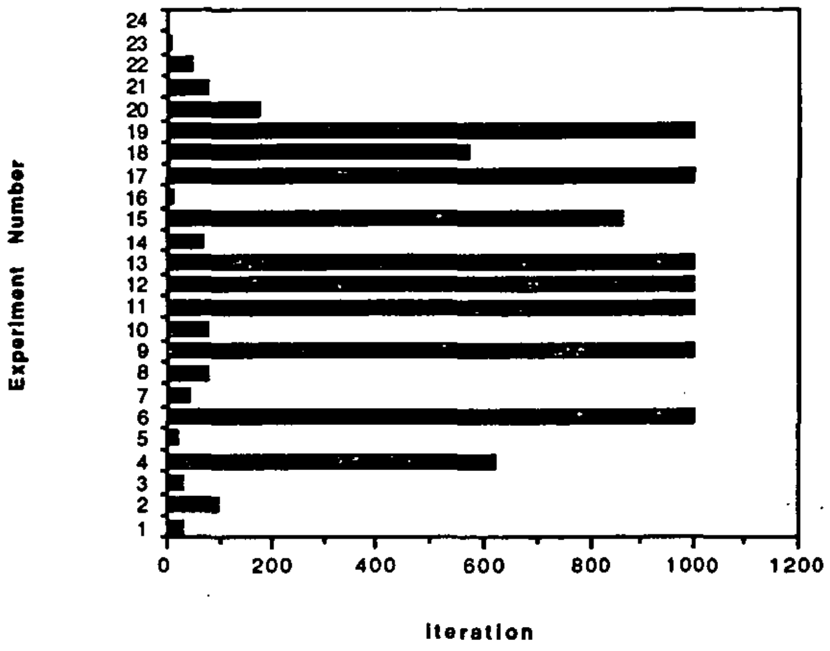
Experiment	Mean	Variance
2 output case	0.256	0.752
3 output case	0.350	0.852
4 output case	1.527	2.053
5 output case	3.440	3.225
6 output case	10.388	6.110
7 output case	21.272	8.913
8 output case	32.861	9.040

Table F7.1 Mean and variance of the sum squared error after 1000 iterations of the reduced input set, three layer quantisation neural network structure

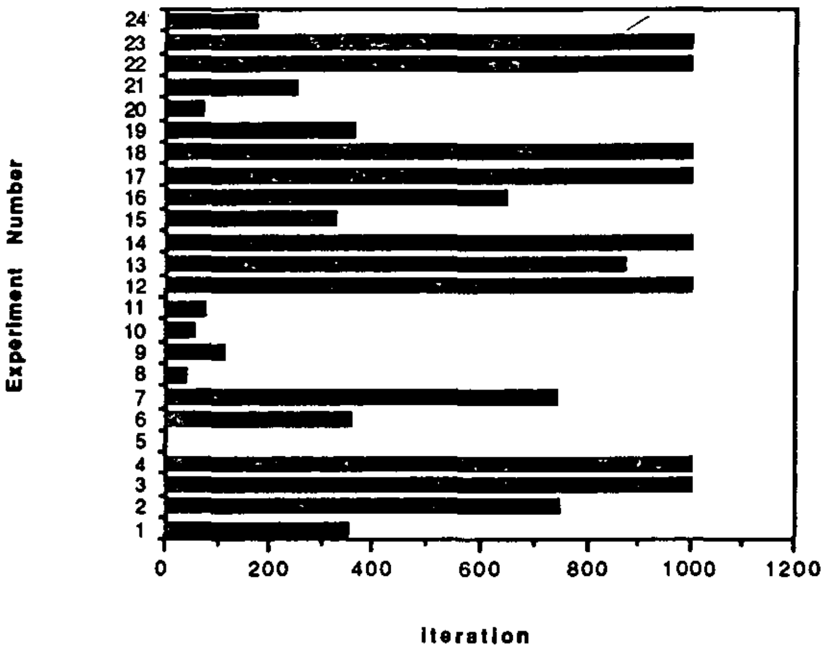
Analysis:

The larger neural network structure is better able to model quantisation transformation, whether it is linear or non linear. This explains why the sum squared error convergence performance of the multilayered quantisation system (see appendix F3) is much better than that of the single layered system. The reduced data sets allow the simple linear transformations to be more probable. Therefore a multilayer quantisation neural network structure with a reduced training data set is more likely to converge than a single layer quantisation neural network structure with a large training set.

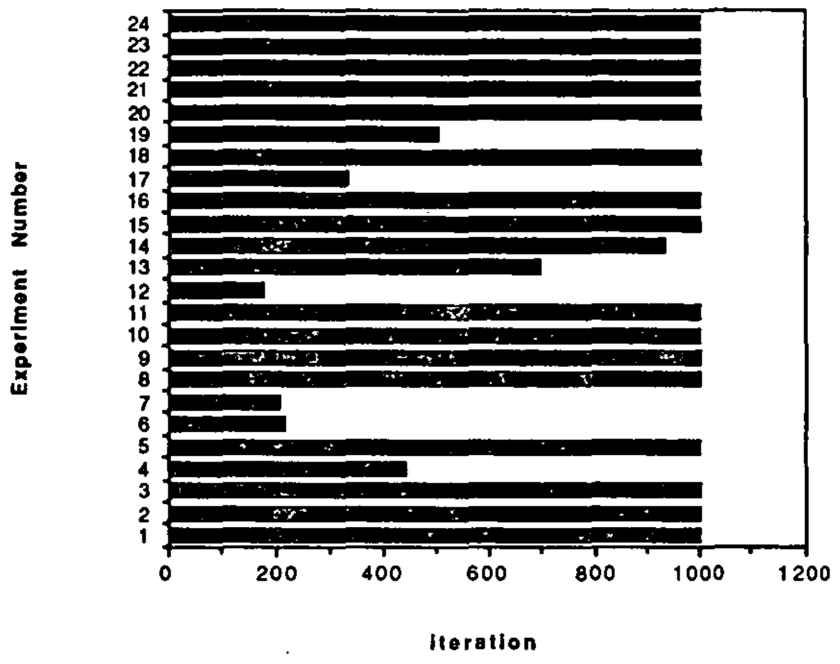
Graphs:



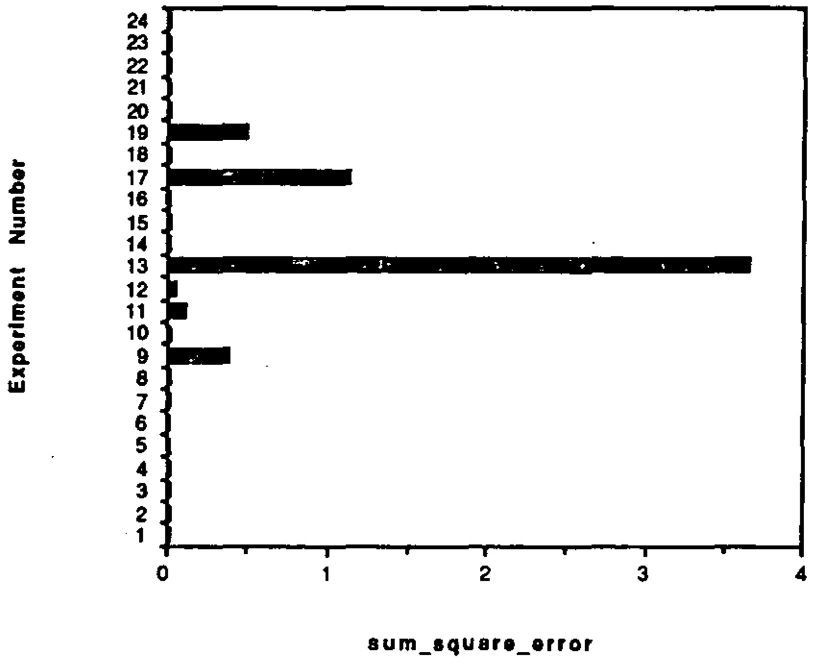
Graph F7.1 a. Number of iterations required for convergence of the reduced input set three layer quantisation neural network structure with two input and hidden nodes



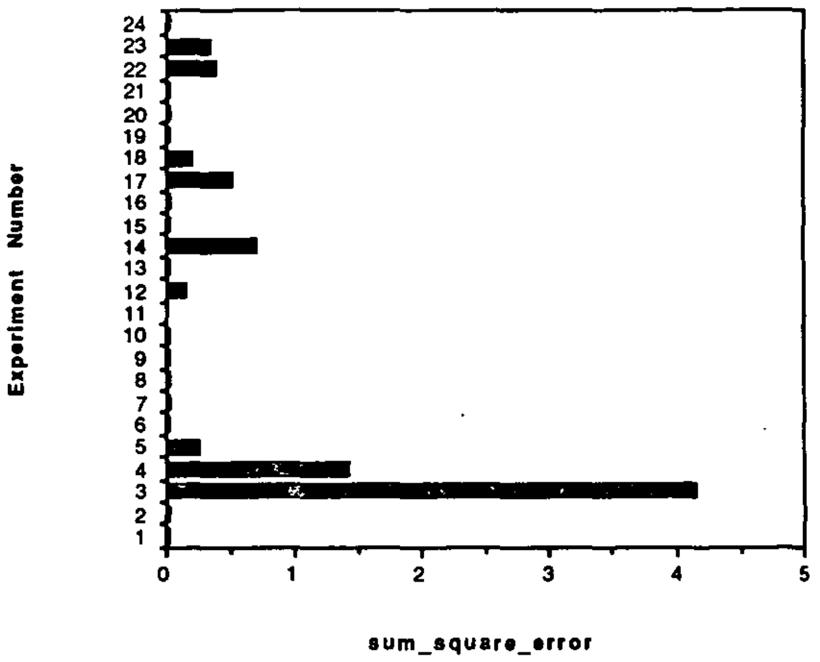
Graph F7.1 b. Number of iterations required for convergence of the reduced input set three layer quantisation neural network structure with three input and hidden nodes



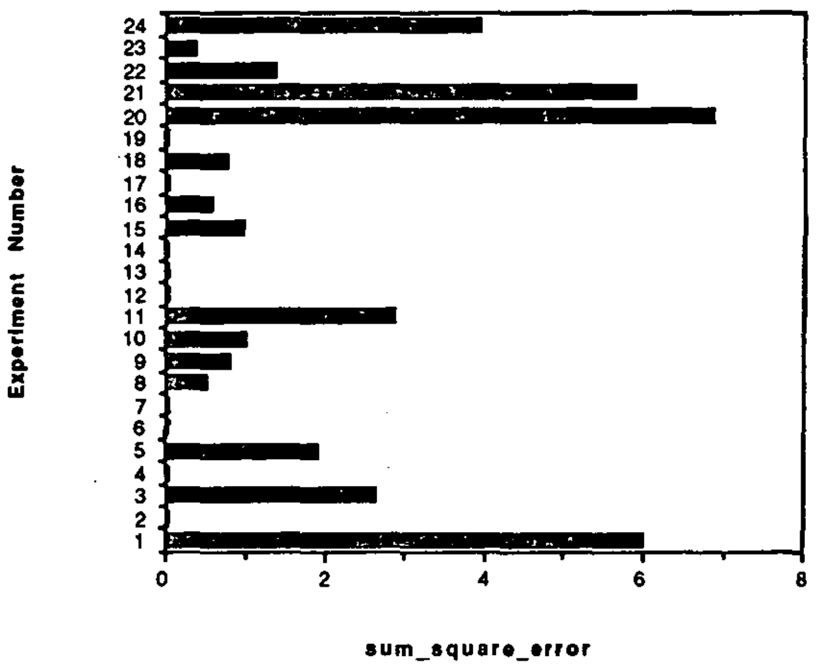
Graph F7.1 c. Number of iterations required for convergence of the reduced input set three layer quantisation neural network structure with four input and hidden nodes



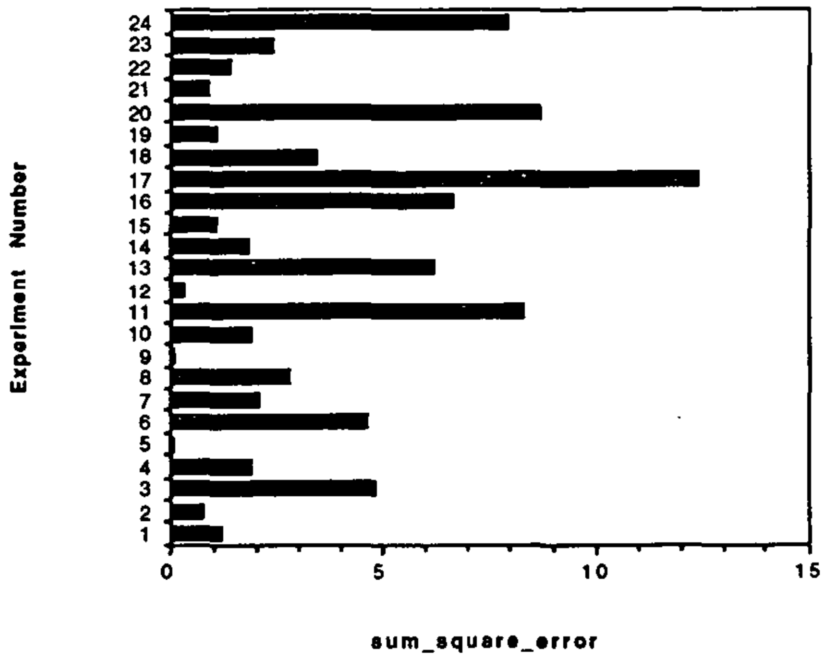
Graph F7.2 a. Sum square error performance of the reduced input set three layer quantisation neural network structure with two input nodes after 1000 iterations



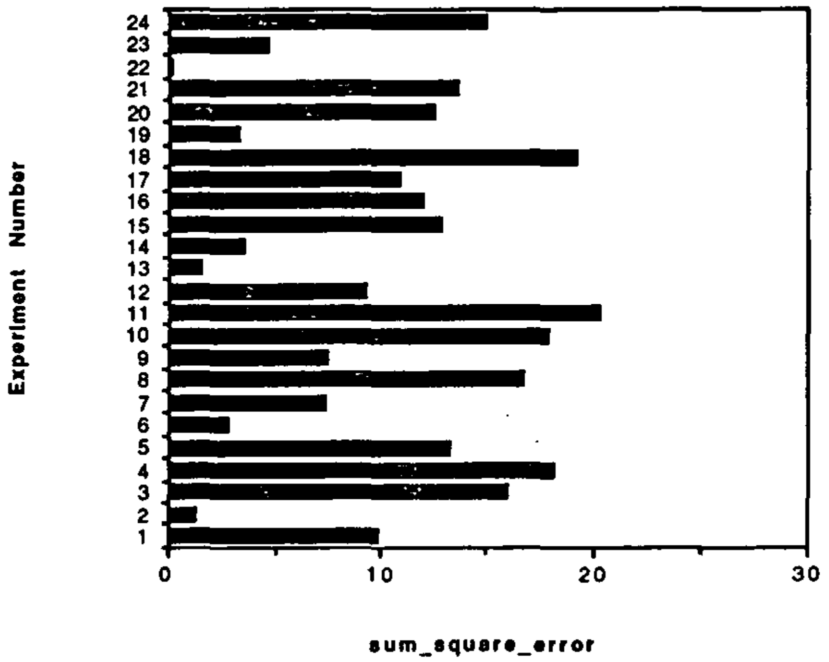
Graph F7.2 b. Sum square error performance of the reduced input set three layer quantisation neural network structure with three input nodes after 1000 iterations



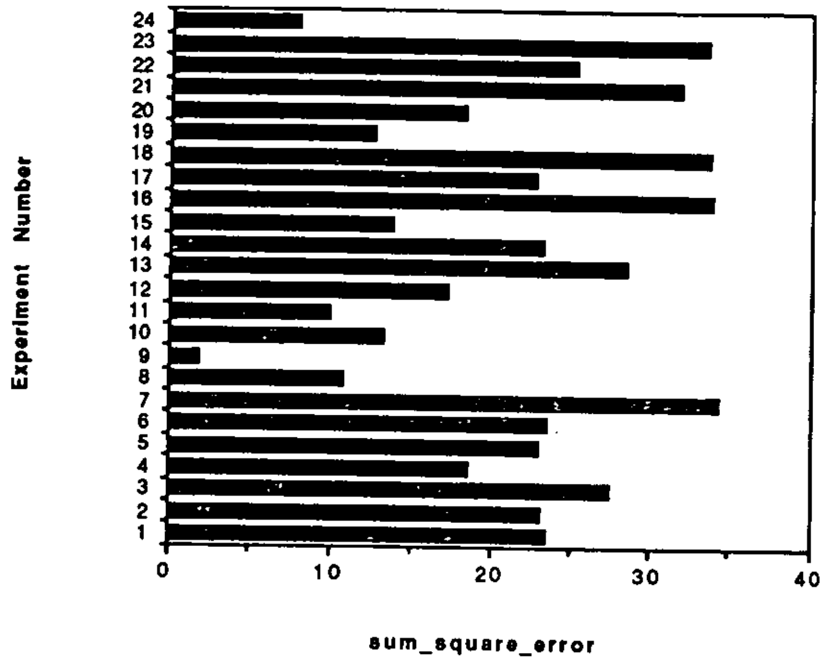
Graph F7.2 c. Sum square error performance of the reduced input set three layer quantisation neural network structure with four input nodes after 1000 iterations



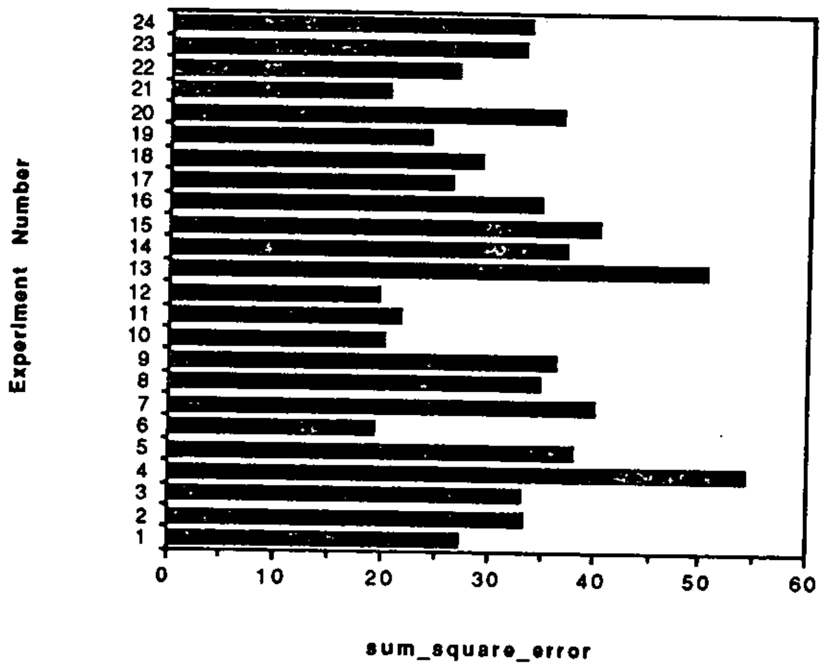
Graph F7.2 d. Sum square error performance of the reduced input set three layer quantisation neural network structure with five input nodes after 1000 iterations



Graph F7.2 e. Sum square error performance of the reduced input set three layer quantisation neural network structure with six input nodes after 1000 iterations



Graph F7.2 f. Sum square error performance of the reduced input set three layer quantisation neural network structure with seven input nodes after 1000 iterations



Graph F7.2 g. Sum square error performance of the reduced input set three layer quantisation neural network structure with eight input nodes after 1000 iterations

Appendix F8. Reduced Input Quantising and Decoding Experiments

Aim:

To identify the ability of three layers of hidden nodes to model the arbitrary quantisation and decoding of a real valued input to a real valued output over a reduced training set.

Method:

The data sets were generated by a seeded pseudorandom scheme. The data was generated in an identical manner to that of appendix F4. The reduced data sets were defined in an identical manner to that of appendix F5.

The initial neural network configurations where generated in the same manner as those of appendix A1. The quantisation nets with one input node three layers of n hidden nodes and one output node are defined by $2*n^2 + 5*n + 1$ weight values.

The experiments were generated with a pseudorandom scheme. Twenty four random neural network configurations were selected to be trained on random data sets. The temperature value of 0.1 and a learning rate of 0.01 was used. The experiments were run for 1000 iterations. The sum squared error convergence characteristics are shown in graphs F8.1 a- d and table F8.1.

Results:

The sum squared error values of a multilayered quantising and decoding neural network structure trained on reduced data sets increase as the number of inputs increase.

The sum squared error convergence performance is much improved over the reduced training sets $n \geq 5$. Table F8.1 shows the mean and variance of the sum squared error performance for the specific number of output nodes. Graphs F8.1 a- d show the sum squared error performance for each of the experiments.

The sum squared error values of this experiment are better than those of the single layered decoding experiment as discussed in appendix F6.

Experiment	Mean	Variance
5 hidden nodes	15.885	7.525
6 hidden nodes	27.288	12.472
7 hidden nodes	38.441	18.990
8 hidden nodes	53.345	22.789

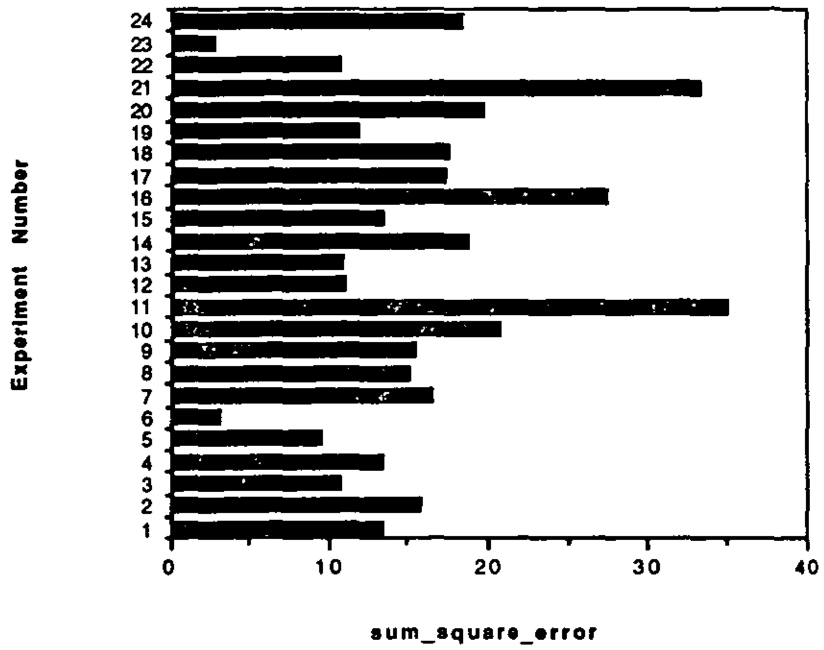
Table F8.1 Mean and variance of the sum squared error after 1000 iterations of the reduced input set, four layer quantisation and decoding neural network structure

Analysis:

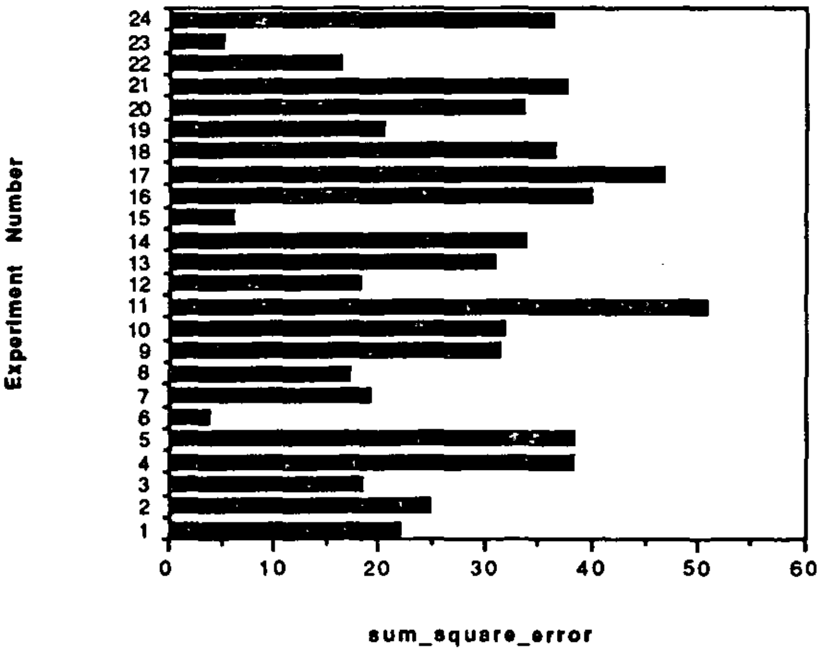
The reduced data sets allow simpler linear transformations to be more probable and so are easier to model than larger training sets. This is demonstrated by the improved performance of the training algorithms for these reduced training sets.

The improved representational power of larger neural network structures over smaller neural network structures is demonstrated by the smaller sum squared error values of this experiment as compared to that of the single layered decoding neural network of appendix F6.

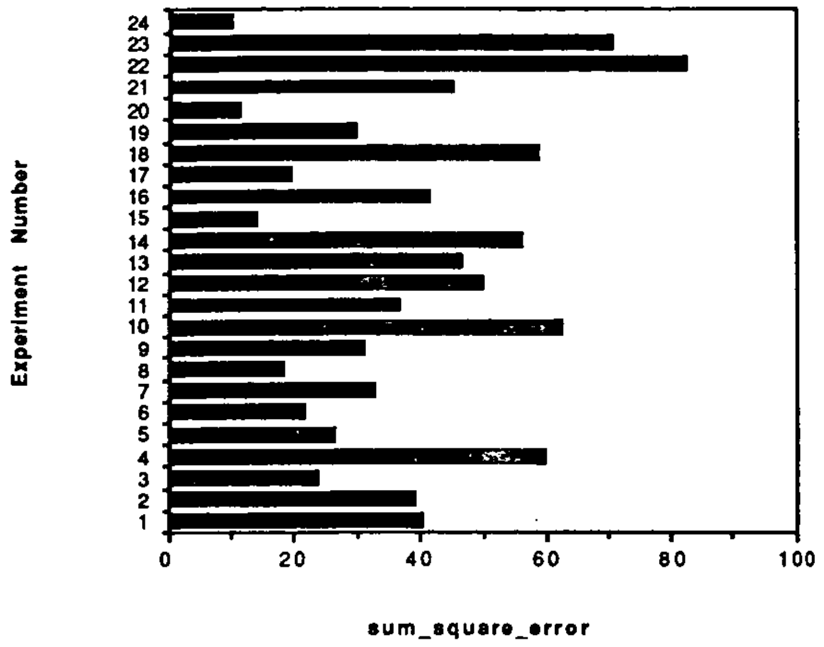
Graphs:



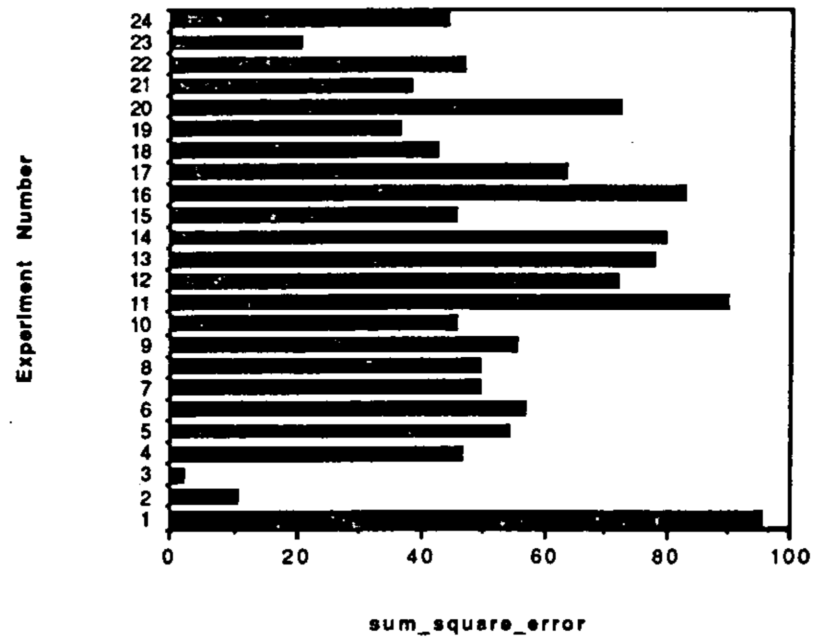
Graph F8.1 a. Sum square error performance of the reduced input set four layer quantisation and decoding neural network structure with five nodes in each hidden layer(1000 iterations)



Graph F8.1 b. Sum square error performance of the reduced input set four layer quantisation and decoding neural network structure with six nodes in each hidden layer(1000 iterations)



Graph F8.1 c. Sum square error performance of the reduced input set four layer quantisation and decoding neural network structure with seven nodes in each hidden layer(1000 iterations)



Graph F8.1 d. Sum square error performance of the reduced input set four layer quantisation and decoding neural network structure with eight nodes in each hidden layer(1000 iterations)

Appendix G: Training a Neural Network Controller for Adhesive Dispensing

Aim:

To examine various methods of training a neural network controller for adhesive dispensing (see chapter nine, Williams et al '90 and West '92). To examine the performance of the learning phase and to evaluate the performance of the execution phase. To examine the ease with which the trained neural network's structure and behaviour could be understood and explained.

Method:

Eight different training experiments were implemented corresponding to the different training data, neural network structure and weights that were assigned.

Data sets

The following training data were used;

i. Real control data:

These were obtained by observing the adhesive dispensing system under control with a rule based bang bang controller. The problem with using this type of data is that all the possible process faults may not be present in the data.

ii. Hand crafted data:

All the observed process characteristics and process faults were used to construct a set of training data. This set had the advantage of possessing all the process characteristics and faults that had been observed in many experiments. Also this training set was small since only useful learning data had been included. This disadvantage with this data set is that any hidden properties of the real system such as correlations between the system variables were lost in the hand crafted data.

Neural network structure

The following neural network structures were used;

i. Fully connected:

This neural network structure consisted of the seven input nodes and the ten output nodes specified by the system. The fully interconnected structure has ten hidden nodes. These hidden nodes corresponded to the three banded regions of the area, area_change and pulse_height system variables (six hidden nodes) and the four threshold boundaries of the rise_time, fall_time, pulse_width and box_area_ratio system variables (four hidden nodes).

ii. Structured fully interconnected neural network:

This neural network structure consisted of the specified input and output nodes, corresponding to the system variables and six hidden nodes. The threshold units of the four threshold boundaries of the rise_time, fall_time, pulse_width and box_area_ratio system variables can be implemented without hidden nodes while the six hidden nodes correspond to the three banded regions of the area, area_change and pulse_height system variables (six hidden nodes). The connection pattern of this system was;

a. All the input nodes were fully interconnected to all the hidden nodes.

b. All the input nodes were fully interconnected to all the thresholding flag outputs, namely the rise_time_flag, fall_time_flag, pulse_width_flag and box_area_ratio_flag nodes.

c. All the hidden nodes were fully interconnected to all the banded flag and decision outputs, namely the area_action,change_area, bubble_flag, bubble_decision, pulse_height_flag and pulse_height_decision nodes.

iii. Structured partially connected neural network:

This neural network structure consisted of the specified input and output nodes, corresponding to the system variables and six hidden nodes. The threshold units of the four threshold boundaries of the rise_time, fall_time, pulse_width and box_area_ratio system variables can be implemented without hidden nodes while the six hidden nodes correspond to the three banded regions of the area, area_change and pulse_height system variables (six hidden nodes). The connection pattern of this system was;

a. Each thresholded input node was connected to its corresponding output flag and no other, e.g. the box_area_ratio input node was connected to the box_area_ratio_flag output node.

b. The banded input nodes were connected to a pair of hidden nodes and no others, e.g the area input node was connected to midnode1 and midnode2 and no others.

c. Each hidden node pair was connected to its corresponding output flag or decision node, e.g. the midnode1 and midnode2 nodes were connected to output nodes area_action and change_area and no others.

Neural network weights

The following weight specifications were used;

i. Small random weight values:

This means that no information is given to the initial neural network structure.

ii. Small hand crafted weights:

This means that the knowledge available about the control problem is used to give the neural network structure an approximate measure of where the set points of the banded region and thresholds are in the input space.

iii. Large derived weights:

This means that the greater knowledge about the control problem is used to specify the set points of the banded region and thresholds are in the input space as well as the tolerances that are required on these set points. The smaller the tolerances that are required the greater the magnitude of the weights of the neural network. This is also related to the temperature value of the neural network.

Experiments

The neural network convergence experiments were run with a learning rate of 0.01 and a temperature of 0.1. The maximum number of iterations that were allowed was 5000. This set an upper limit on the training procedure. If the training algorithm did not converge within this limit the sum squared error values after 5000 iterations were used to examine the performance of the training algorithm.

Results:

The results of the experiments are shown in table G.1. The sum squared error convergence characteristics of each of the experiments are shown in graphs G.1 a- f and G.2 a- f. The performance of the converged neural network controller on the real process is shown in graph G.3.

Neural Network Structure	Initial Weight Specification	Simulated Data		Real Data	
		Iterations	Sum square error	Iterations	Sum square error
Fully Interconnected Structure	Small Random Weights	5000	155.957	5000	279.987
	Small Specified Values	5000	20.079	5000	269.436
Reduced Interconnected Structure	Small Random Weights	2343	0.045	5000	146.888
Minimal Designed Structure	Small Random Weights	5000	41.065	5000	159.561
	Small Specified Values	2874	0.187	5000	123.111
	Large Derived Weights	4	0.065	44	0.116

Table G.1 Convergence properties of the training algorithm for different initial neural network structure and training data

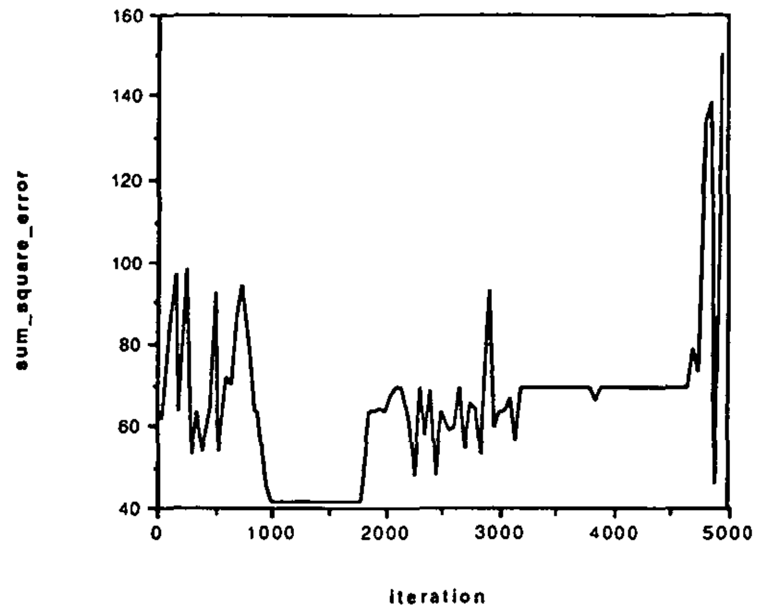
Analysis:

Improved sum squared error convergence performance was obtained by increasing the design effort implementing the neural network controller. The most significant factor that effected the performance of the automated training was the structure of the initial neural network that was trained. Pruning the neural network to the minimum that was required to model the problem aided optimal convergence.

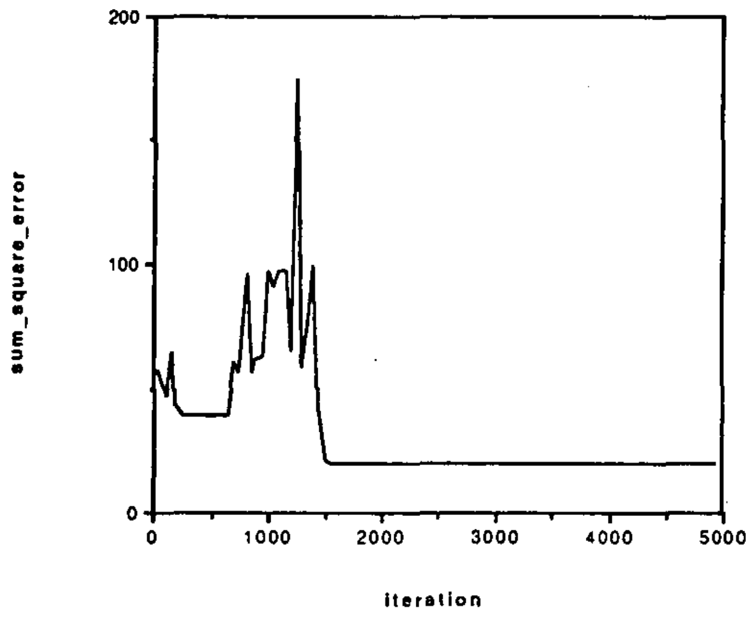
The second significant factor was the initial weight specification. Small random weights provided no information. The sum squared error convergence was not good. Including small weight values that were designed to place the initial neural network in the approximate region of the control problem set points improved the convergence performance. This model converged on the simulated data but did not converge within the 5000 iterations (examined in this experiment) on real data. This was due to the fact that the real training data contained data that required decisions to be made over small tolerances which can only be achieved by a neural network with a low temperature or one with large weight values.

Deriving large weight values that modelled the control set points to finer tolerances ensured that the automated training would converge in fewer iterations. This method effectively starts the neural network in a configuration that is very close to a suitable controller. The automated training fine tunes the neural network to produce the required controller. The fine tuning is normally necessary given real data since the real data will possess properties that are not modelled by the designed neural network.

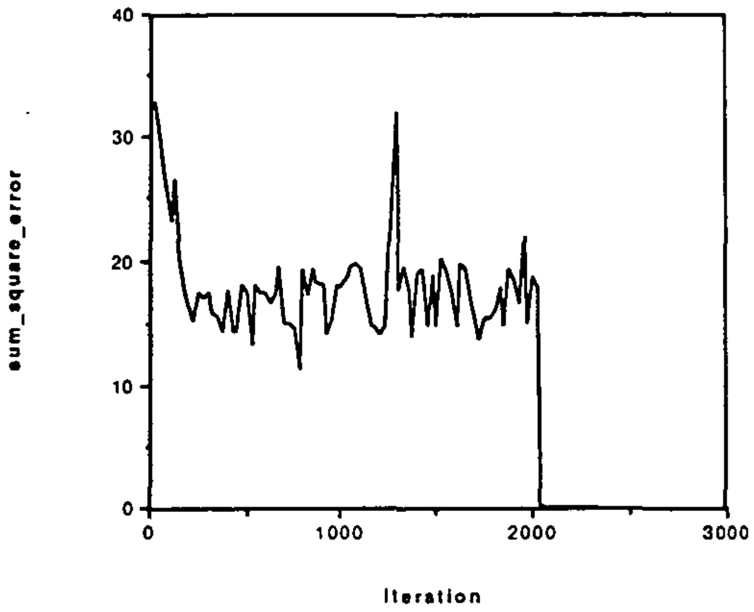
Graphs:



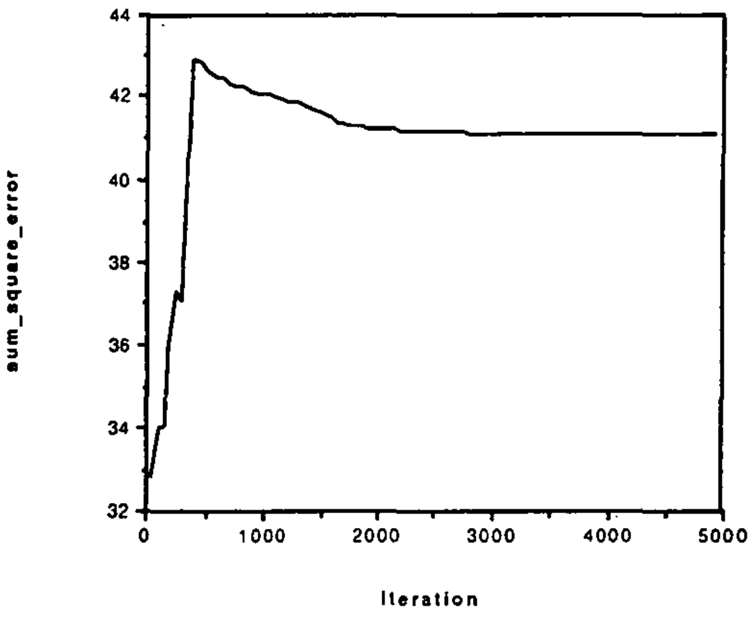
Graph G.1 a. Sum squared error performance of the fully interconnected neural network structure with small random weights trained on selected simulated data



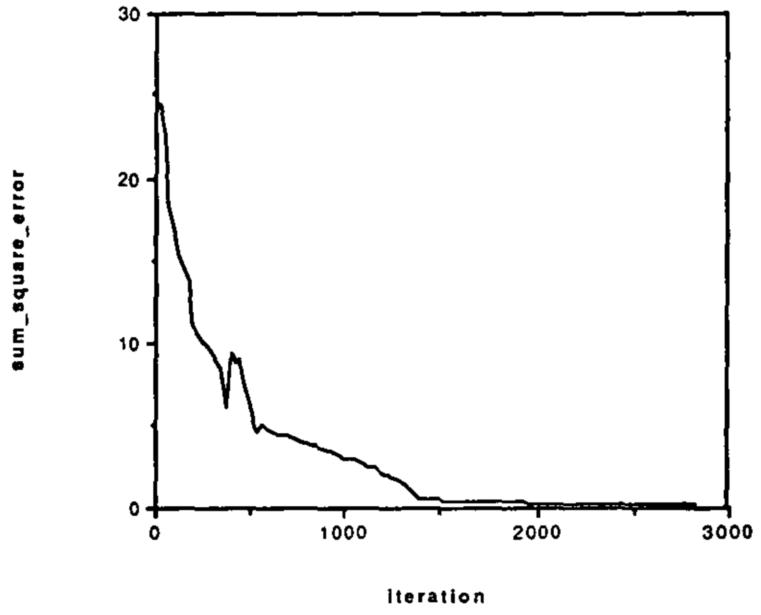
Graph G.1 b. Sum squared error performance of the fully interconnected neural network structure with small specified weight values trained on selected simulated data



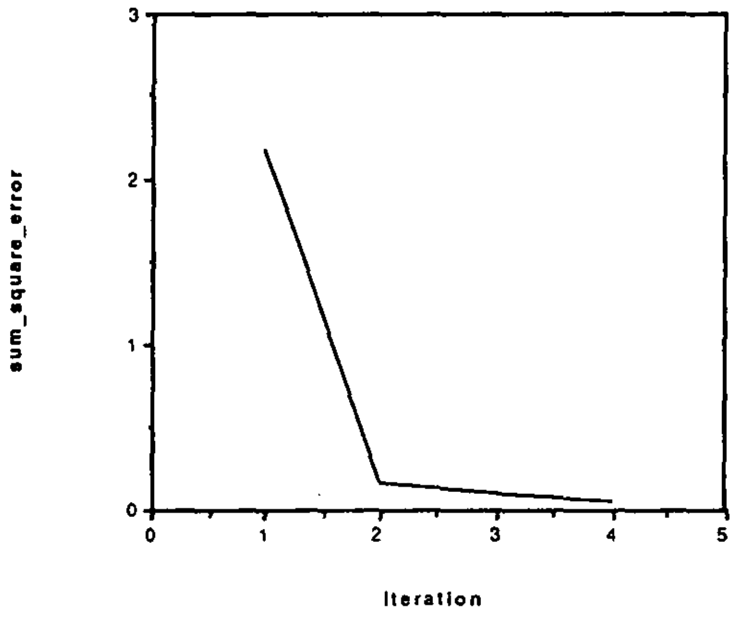
Graph G.1 c. Sum squared error performance of the reduced interconnected neural network structure with small random weights trained on selected simulated data (note the change of scale on the horizontal axis)



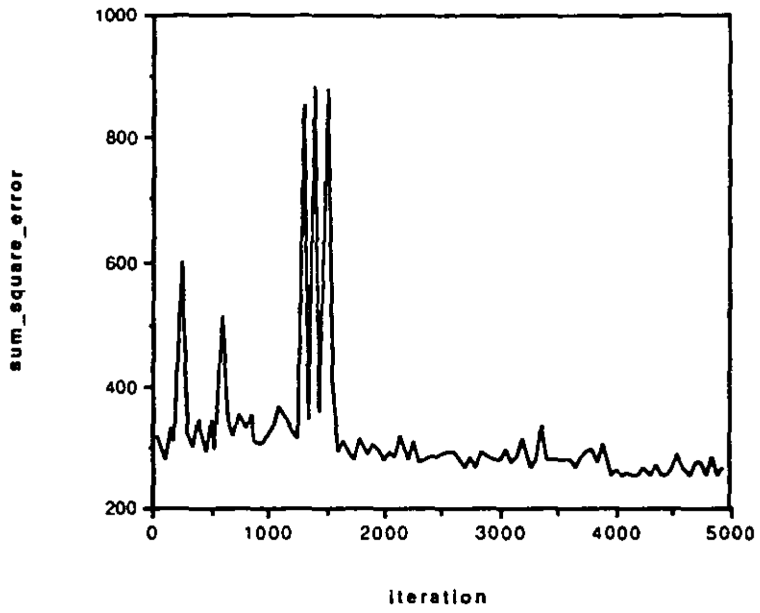
Graph G.1 d. Sum squared error performance of the minimal designed neural network structure with small random weights trained on selected simulated data



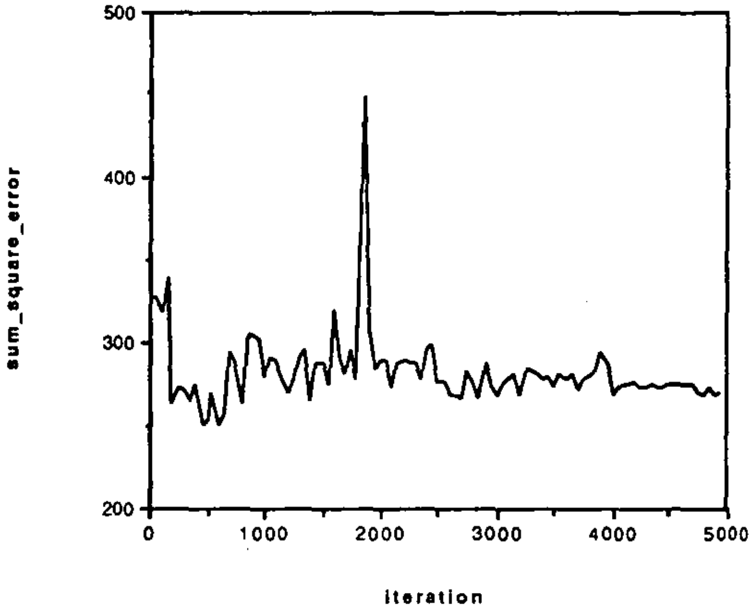
Graph G.1 e. Sum squared error performance of the minimal designed neural network structure with small specified weight values trained on selected simulated data (note the change of scale on the horizontal axis)



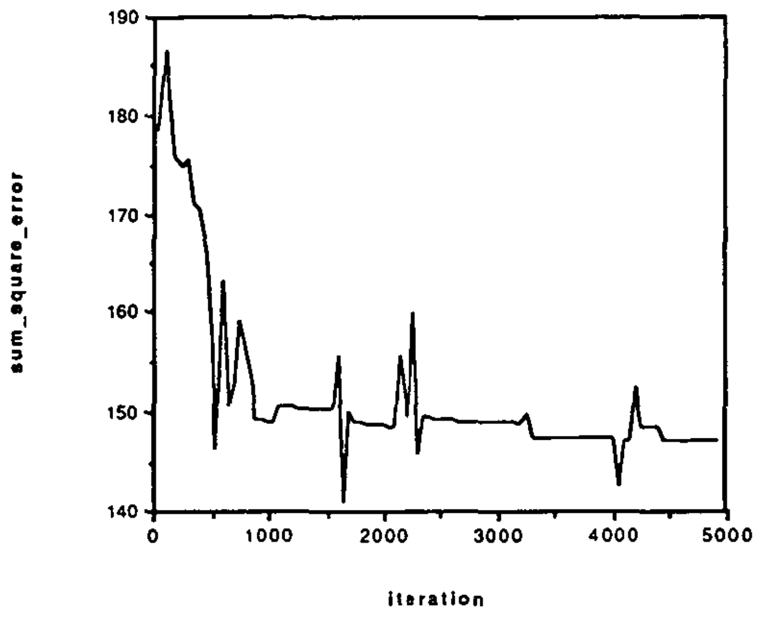
Graph G.1 f. Sum squared error performance of the minimal designed neural network structure with large derived weight values trained on selected simulated data (note the change of scale on the horizontal axis)



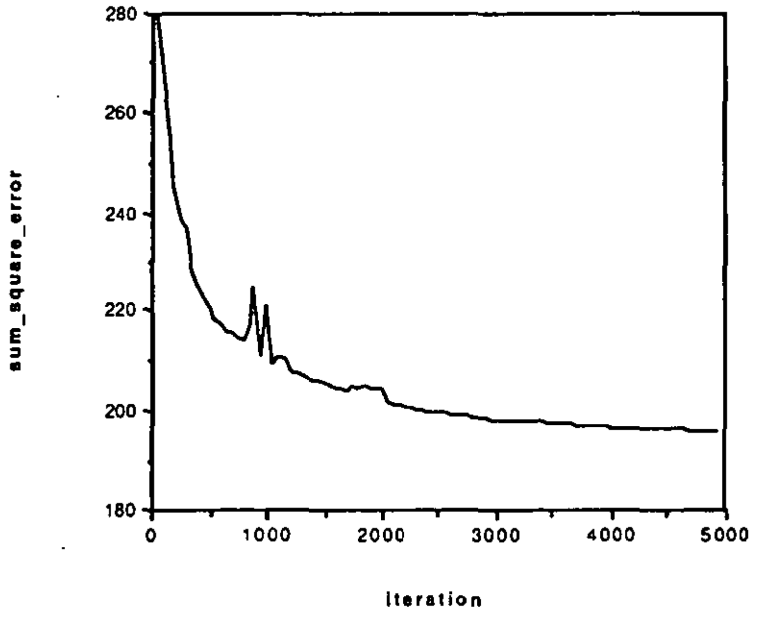
Graph G.2 a. Sum squared error performance of the fully interconnected neural network structure with small random weights trained on selected real data



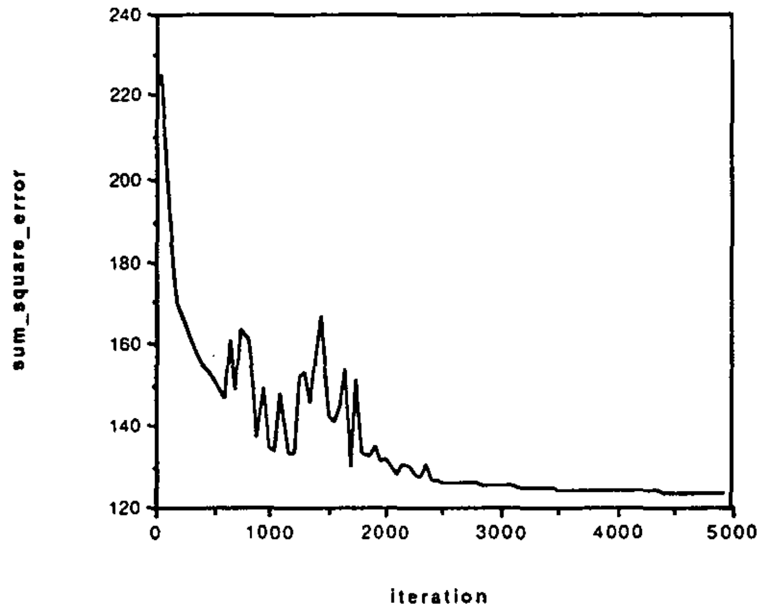
Graph G.2 b. Sum squared error performance of the fully interconnected neural network structure with small specified weight values trained on selected real data



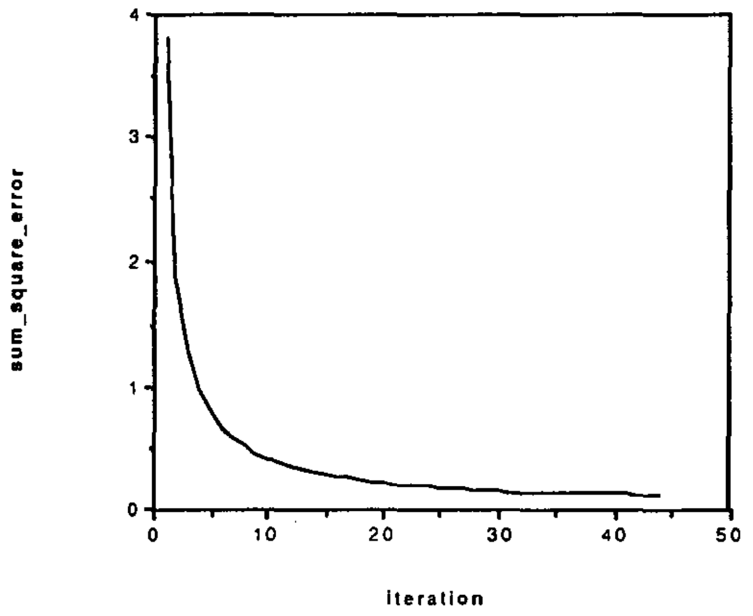
Graph G.2 c. Sum squared error performance of the reduced interconnected neural network structure with small specified weight values trained on selected real data



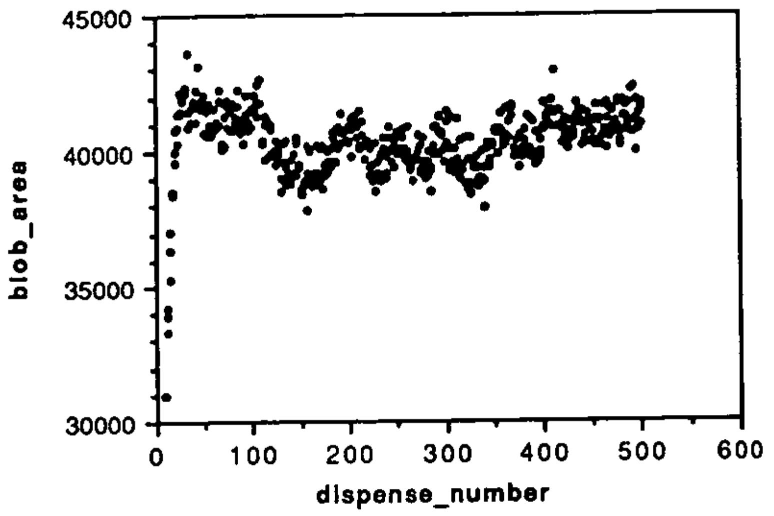
Graph G.2 d. Sum squared error performance of the minimal designed neural network structure with small random weights trained on selected real data



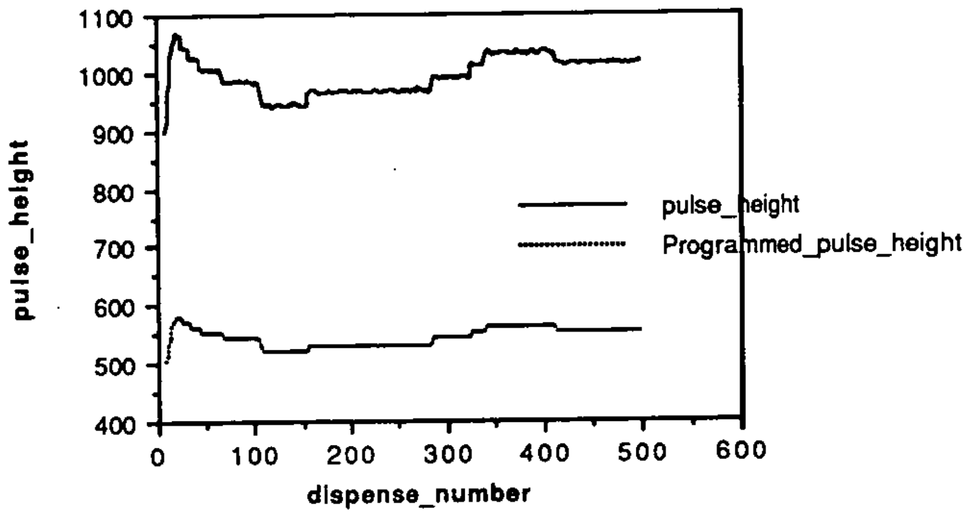
Graph G.2 e. Sum squared error performance of the minimal designed neural network structure with small specified weight values trained on selected real data



Graph G.2 f. Sum squared error performance of the minimal designed neural network structure with large derived weight values trained on selected real data (note the change of scale on the horizontal axis)



Graph G.3 a. Blob area control performance of a neural network controller. Note that the blob area variation is kept within the 5% (2000 units) limits of the target value (40000 units)



Graph G.3 b. The applied pressure pulse variation used to produce the control performance of Graph G.3a. Note the initial increase in pressure required to reach the target area of 40000 units when the system initially started dispensing blobs of 30000 units