A Study of the Design and Analysis of Feed Forward Neural Networks

by

Graham Paul Fletcher

B. Sc. (HONS).

A Doctoral Thesis
Submitted in Partial Fulfilment of the Requirements
For the Award of Doctor of Philosophy
of Loughborough University of Technology
1995

Declaration

I declare that this thesis is a record of research work carried out by me, and that this thesis is my own composition. I also certify that neither this thesis nor the original work contained therein has been submitted to this or any other institution for a higher degree.

Graham Paul Fletcher

Acknowledgements

I wish to express my gratitude to my parents, my bank manager and the engineering and physical sciences research council who's joint sponsorship has made this research possible.

I would also like to thank my supervisor Dr C.J.Hinde for his help, advice and, most importantly, the training on how to survive in academia. Thanks also to the many other staff at the university who have helped with either suggestions or employment.

Finally the moral support and proof reading provided by my family and friends has been instrumental in helping me finish this thesis.

Abstract

This thesis shows that a design and analysis system for feed forward neural networks is desirable, and that the currently available techniques do not work. Methods have been presented that solve the problem of analysis, showing that analysis is possible and desirable for classification networks.

The biggest limitation is the size of the network and that the analysis tools are only applicable to properly designed classification systems. A method of reducing the size of classification networks is presented along with a design methodology for non classification systems.

CONTENTS

LITERATURE REVIEW
CHAPTER 1

Outline Of Chapter

The aim of this chapter is three fold. Firstly it is to provide an introduction to artificial neural networks in sufficient detail to allow a non specialist access to the work contained in the later chapters. Secondly this chapter maps out the development of artificial neural networks by charting the major contributions to the field. Lastly, and most importantly, the chapter contains several of my own interpretations of the history, and predictions about the current and possible future directions of research.

The chapter is split into three major sections.

Section 1. An overview of the biological systems that formed the original neural networks.

Section 2. This section charts the development of feed forward neural models. This model is the best understood of all the current neural models and is the model used for the majority of this thesis.

Section 3. In section three some of the other models are explored.

The Biological Foundations

The term neural network has grown to represent a large number of very different connectionist strategies for solving machine learning and other artificial intelligence problems. Originally they were developed as a method of modelling the neural construction of the brain, in the hope that computers could be built that would emulate the brain's capacity for learning, content addressability and conceptual leap. The starting point for any study of artificial neural networks should begin with a simple understanding of the mechanisms and structures used in biological neural networks. It is these biological systems that provided the original neural models.

The brain is composed of about 10^{11} neurons , figure 1.1 shows a simple neuron. Each neuron consists of the dendrites which are the inputs, the nucleus or processing unit and the axon and synapses which make up the output to the neuron and connect it to the dendrites of the next neuron.

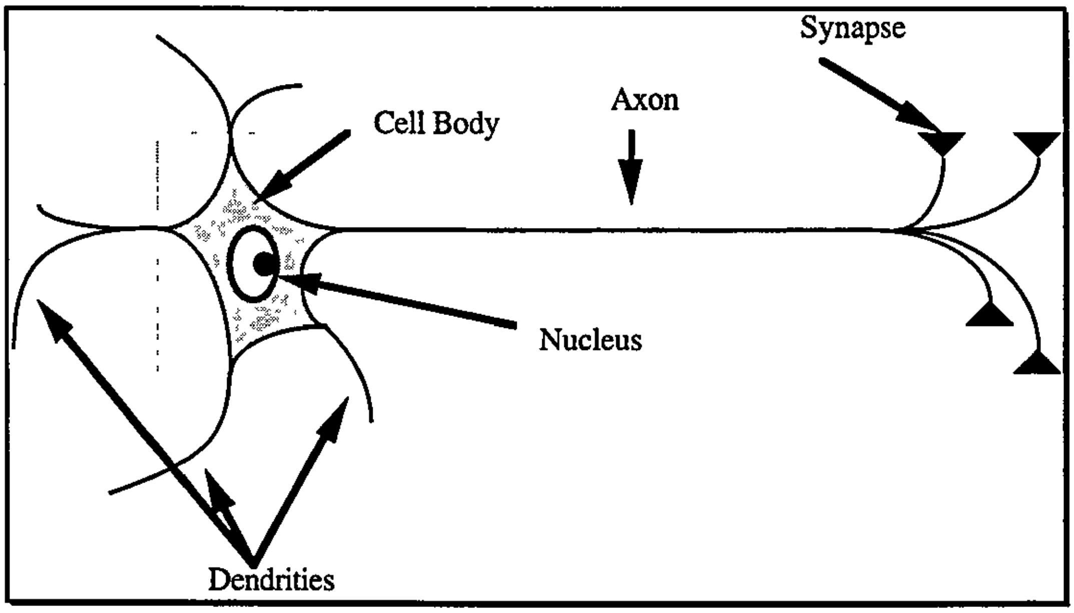


Figure 1.1 A diagram of a simple neuron.

Griffith [1966] provides a simple explanation of the biochemical basis of neurons;

> "The cell is bounded by a surface membrane which is semi-permeable. This means that the membrane will allow certain ions and molecules to pass through it, but not others. The semi-permeability has certain consequences which can be understood using the theory of thermodynamics. One of these is that the interior of the cell normally has a negative electrostatic potential of about -70mV with respect to its

> exterior. However, if this potential difference is artificially altered above a threshold value of about -60mV, the membrane suddenly becomes permeable.... ...This change of permeability is self-propagating, it starts at one point and spreads to adjacent points and so on. Shortly afterwards the membrane recovers its semi-permeability. Thus a transient burst of electrochemical activity is generated."

The membrane of the cell covers the cell nucleus, but also extends down the axons to the synapse. The axons and the synapse form the link between one nerve cell and its neighbours. Griffith further explains

> "Once a cell has fired, the activity passes out to the uttermost extremes (the synapse), which then spew out small quantities of chemicals called transmitters. These pass across narrow gaps to adjacent cells where they alter the permeability of the membranes of those cells."

The potential difference across the cell membrane increases as quantities of the transmitter chemicals are absorbed. However the neuron does not fire until the potential difference reaches -60mV.

The transmission of information around the brain is a complex electro-chemical process, where chemicals are released from the synapses on the sending side of the junctions. This causes the raising or lowering of the electrical potential inside the body of the nucleus. If the potential reaches a threshold a pulse of fixed strength is sent down the axon to the next set of synapses. After firing a cell cannot fire again immediately, but has to wait a fixed length of time called the "refractory period" before it can fire again.

The Development of Feed Forward Models

McCulloch and Pitts [1943] published a now famous paper proposing a simple mathematical model of the neuron, known as the idealised neuron model. Each cell calculated a weighted sum of its inputs and set its output to 1 or 0 depending on whether this sum was above or below a threshold. See figure 1.2 for a schematic representation of the idealised neuron. Formally this is written

$$\text{Output} = f\left(\sum(\text{Input}_n * \text{Weight}_n) - \mu\right)$$

where μ is the threshold

$$f(X) = 1 \text{ if } X \geq 0$$
$$0 \text{ otherwise.}$$

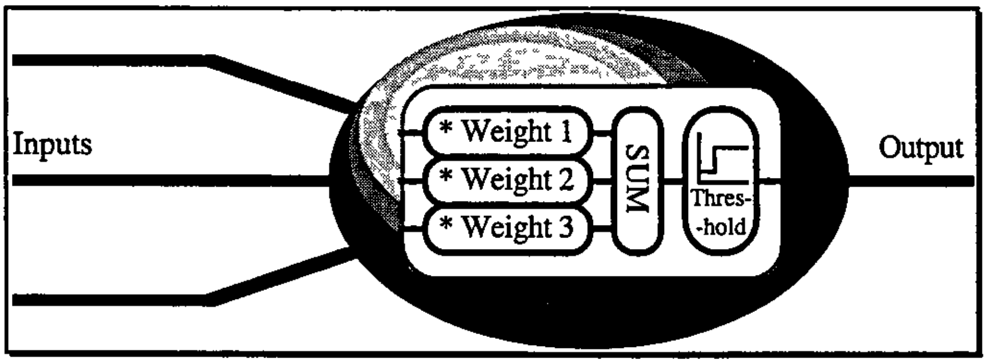


Figure 1.2 A simple idealised neuron

The idealised neuron is a simplification of the real neuron described in the first section. At first glance they appear to be very similar, how do they differ? Figures 1.3 and 1.4 demonstrate the very real gap between the idealised neuron and the neuron described earlier.

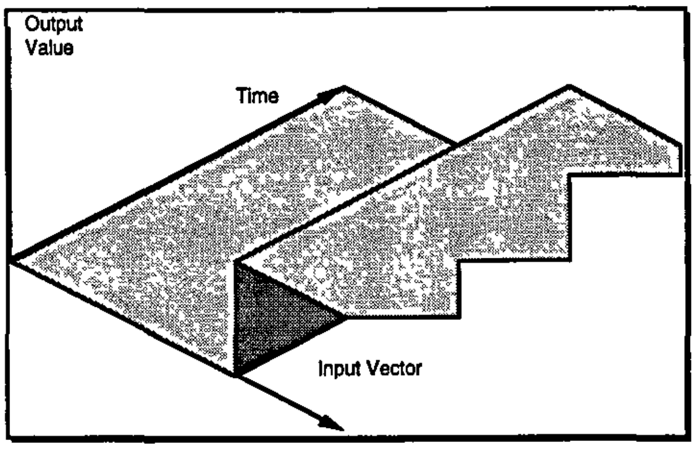


Figure 1.3 The output of an idealised neuron in relation to input value and time. Note that the output value does not change with time.

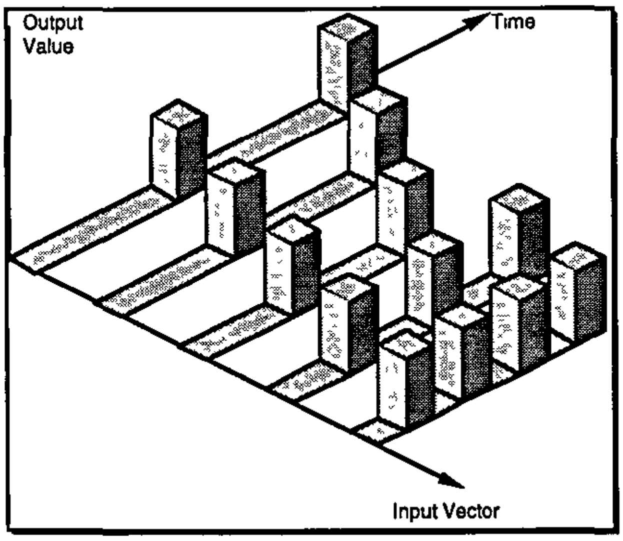


Figure 1.4 A more accurate representation of the activation in real neurons. Note that the output is time dependant.

In a real neuron any value of input will eventually cause it to fire as the neuron stores the chemicals that cause the breakdown of the semi-permeability. A higher input causes a higher frequency of activity as the build-up time between pulses is reduced. By comparison the idealised neuron has no time complexity. A set of inputs will produce an output, this output will not change until the inputs change.

Given the large short fall in complexity the idealised neuron was still a very important mathematical model. It allowed the development of the first training algorithms and analysis techniques.

The simple perceptron model

The most popular use for idealised neurons was the simple perceptron model. The model is a simplification of the full perceptron model proposed by Rosenblatt [1958]. The simpler version allowed the development of training algorithms, these algorithms were not extensible to the full model. Consequently, Rosenblatt's full model was never popular, mainly due to its over complexity. Figure 1.5 shows a simple perceptron model.

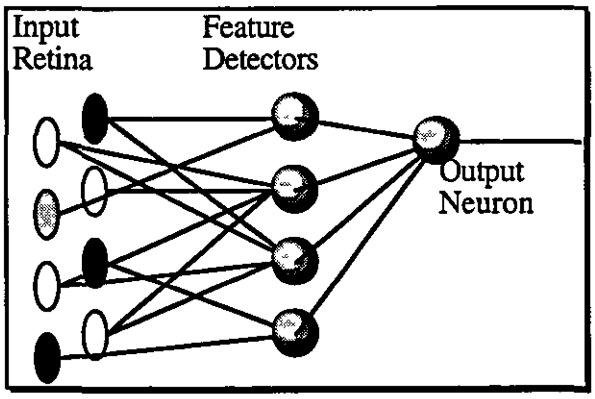


Figure 1.5 The simple perceptron model.

In the simple model all cells output a binary value, there are no connections between cells of the same layer and only the weights between the feature detectors and the output neuron are modified during training. This means that the feature detectors are not modifiable and so correspond to **fixed** feature sensors. In the simple model the input retina is normally left out and the feature sensors referred to as the input cells.

The simple perceptron model cuts the possible inputs into two categories. Those that make it return true and those that don't. Therefore the model is performing a set membership function that identifies a subset of its inputs as members.

The border between these categories, known as the decision plane, is built up using the decision planes of the output and feature detector idealised neurons. The decision plane of an idealised neuron is always a straight line in D dimensions when the neuron has D inputs.

The development of a simple training algorithm by Rosenblatt [1962] is, in my opinion, the single most important step in the development of artificial neural networks. Although his work has now been far superseded it was his ideas that caused the first explosion in connectionist research. The aim of the training algorithm was to calculate a set of weights for the connections between the feature detectors and the output node that allowed correct classification of a set of training examples. For example the decision plane shown in figure 1.6 could have been calculated as a response to the example points.

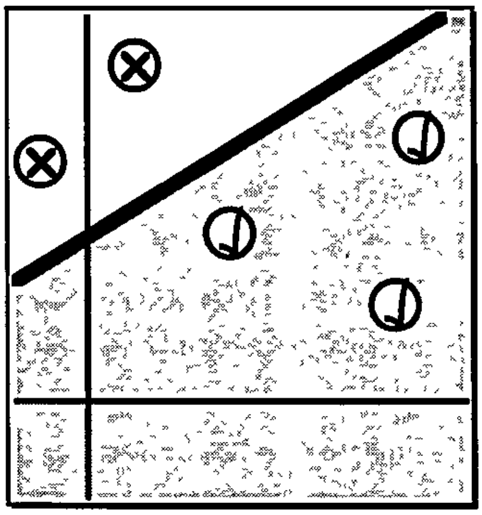


Figure 1.6 A possible decision plane and training examples for the simple perceptron model.

The perceptron model learns by means of a supervised learning process. In other words it is necessary to supply the correct classification for each example input. For each training example the required output is D, and the actual output is R.

If R = D then the perceptron has made the correct decision and no changes need to be made. However if R $\neq$ D then some changes need to be made so as to correct the error. An example is made up of the values supplied by the fixed feature detectors and the required output value D. If the following changes are made to the input weights in turn then the decision plane will have moved.

$$W_n^{new} = W_n^{old} + \Delta W_n$$

$$\Delta W_n = -\partial(R \,.\, Input_n)$$

The algorithm is repeated until all the training examples give the correct classification. ∂ is a small positive number that dictates the rate of convergence on the answer. If ∂ is too small then a large number of iterations will be required, but if it is too big then the algorithm will become unstable and fail to converge. The choice of correct value for ∂ can be difficult and requires some insight. The diagram below shows a simple training sequence.

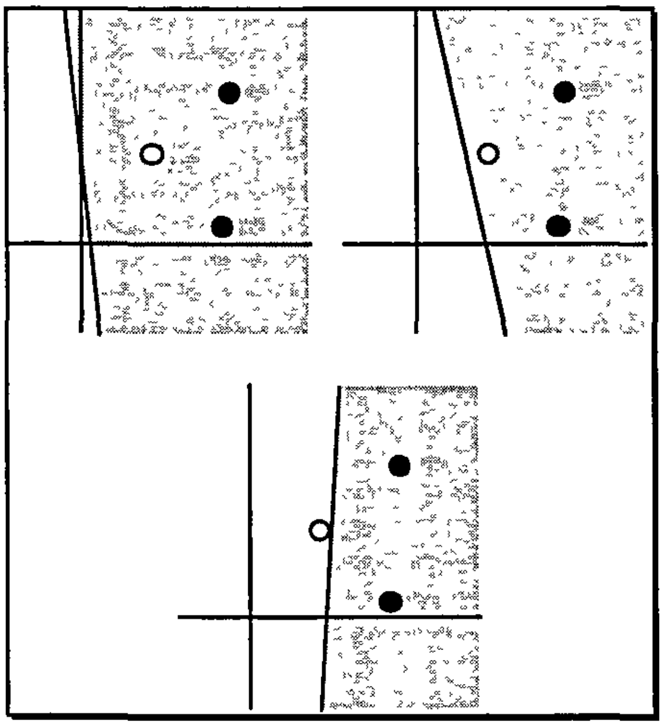


Figure 1.7 The result after each training iteration when the perceptron convergence algorithm is used with a simple problem.

Limitations Of the simple perceptron model

"Perceptrons: An introduction to Computational Geometry" [Minsky & Papert, 1969] provided the most complete mathematical treatment of perceptrons to that date. The review by Block [1970] states

> " The theory is carefully formulated and focuses on the theoretical capabilities and limitations of these machines... ....They ask novel questions and find some difficult answers".

The most important result exposed the limitations of perceptrons and caused a major shift in research away from neural networks to other forms of artificial intelligence.

Minsky & Papert concentrated their work on analysing the limitations of the simple perceptron model. As the feature detectors are fixed they can be thought of as the inputs to the model. Therefore the simple perceptron model is really a single idealised neuron and the training algorithm for discovering the correct weights for its inputs. The best, and most used, example of the limitations of the idealised neuron was first presented by them in the book. It shows that a simple perceptron network with limited diameter feature detectors cannot recognise whether a figure is connected, that is if all points in a figure are connected or are some separate. The left two examples in figure 1.9 are connected but the right pair are not.

A perceptron network has a limited diameter if each of its feature detectors is only connected to inputs within an area on the retina (figure 1.8).

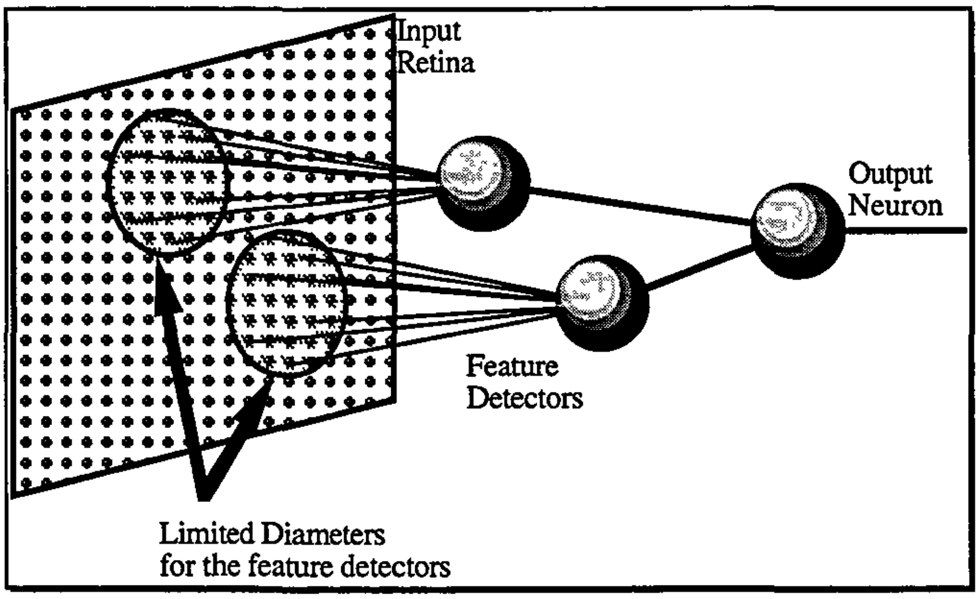


Figure 1.8 A perceptron model with limited diameter feature detectors.

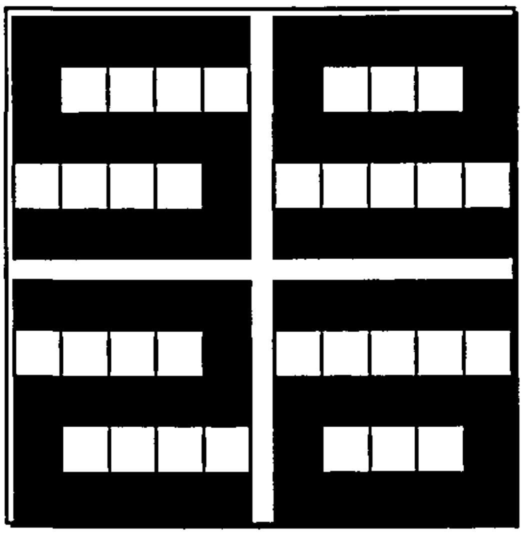


Figure 1.9 The input patterns used to prove the limitations of perceptrons.

Consider the four patterns in figure 1.9 imposed on the retina shown in figure 1.8. If a perceptron network of limited diameter M is to test for connectivity, where M < length of the retina then the conjunctive cells fall into three categories.

1) cells that cover the left hand end of the retina
2) cells that cover the right hand end of the retina
3) cells that cover neither end of the retina

Cells falling into category 3 always see the same input in all four examples and so do not need to be considered as they present no evidence. The cells that view the sides of the retina can see two different patterns. Let these cells differentiate between the patterns returning 1.0 for one of them and 0.0 for the other (figures 1.10 & 1.11).

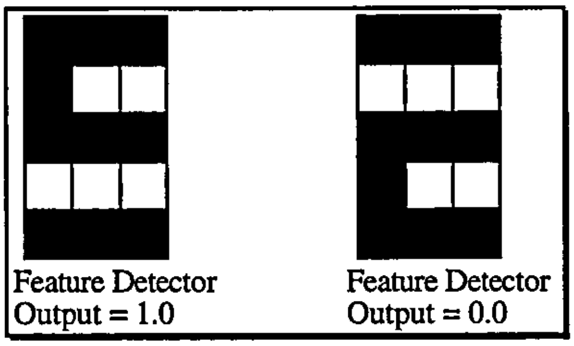


Figure 1.10 The two patterns seen at the left end of the input retina.

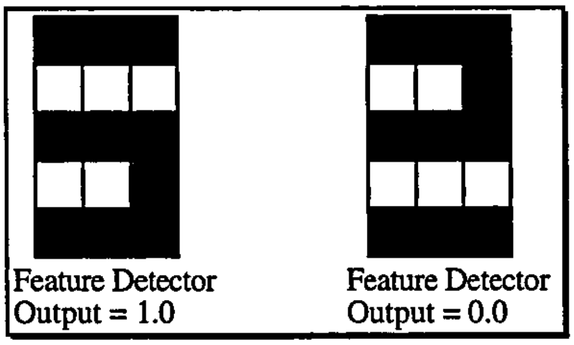


Figure 1.11 The two patterns seen at the right end of the input retina.

The output to the simple perceptron model is calculated by a single neuron. This neuron has two weights and a bias associated with it, let these be W_l, W_r & B. Comparing the required outputs and the values supplied by the feature detectors over the four examples we can achieve four inequalities.

$$1.0\, W_l + 1.0\, W_r - B > O \qquad (1.1)$$

$$1.0\, W_l + 0.0\, W_r - B < O \qquad (1.2)$$

$$0.0\, W_l + 0.0\, W_r - B > O \qquad (1.3)$$

$$0.0\, W_l + 1.0\, W_r - B < O \qquad (1.4)$$

Solving this set of inequalities gives the values of W_l, W_r & B that can correctly calculate the required output function (is the shape connected).

Adding (1.1) and (1.3) $1.0\ W_l + 1.0\ W_r > 2\ B$ (1.5)

Adding (1.2) and (1.4) $1.0\ W_l + 1.0\ W_r < 2\ B$ (1.6)

(1.5) and (1.6) are inconsistent. Therefore the set of inequalities has no solutions and there are no possible values of W_l, W_r & B that can correctly calculate the required output. This means that a perceptron with limited diameter can never test for connectivity.

The full perceptron model removes some of the limitations of the simple perceptron model. In the simple model all cells output a binary output, there are no connections between cells of the same layer and only the input weights to nodes in the last layer are modified during training. This limits the number of training examples that can always be correctly classified.

The full perceptron model

The full perceptron model originally presented by Rosenblatt [1958] contained none of the restrictions listed for the simple perceptron model. This allows much greater freedom in constructing the perceptron. As the cells in layer 1 can now link any weights to the input cells they can compute any conjunction of the input variables in the retina. A similar statement can be said for the cells in layer two; that can compute any disjunction over their inputs.

Any Boolean function can be represented in conjunctive normal form (CNF). Therefore, if the feature recognition neurons are connected to every cell in the retina, then the full perceptron model is capable of representing any Boolean transformation from its input retina to its output cells.

Unfortunately at this time there were no training algorithms for the full perceptron model, making the system nearly useless. This more than any other problem caused a massive reduction in connectionist research between the late sixties and the mid nineteen eighties. However, another major factor was that Minsky and Papert had found all the easy to medium results, leaving little for other people to research.

Enhancements to the simple perceptron model

During the lull in connectionist work there were still some people trying to enhance the simple perceptron model. One of the major problems of the simple perceptron model is that it can only compute linearly separable functions. i.e. There must be a straight line solution between the acceptable and unacceptable inputs. Figure 1.12 shows some non linearly separable functions.

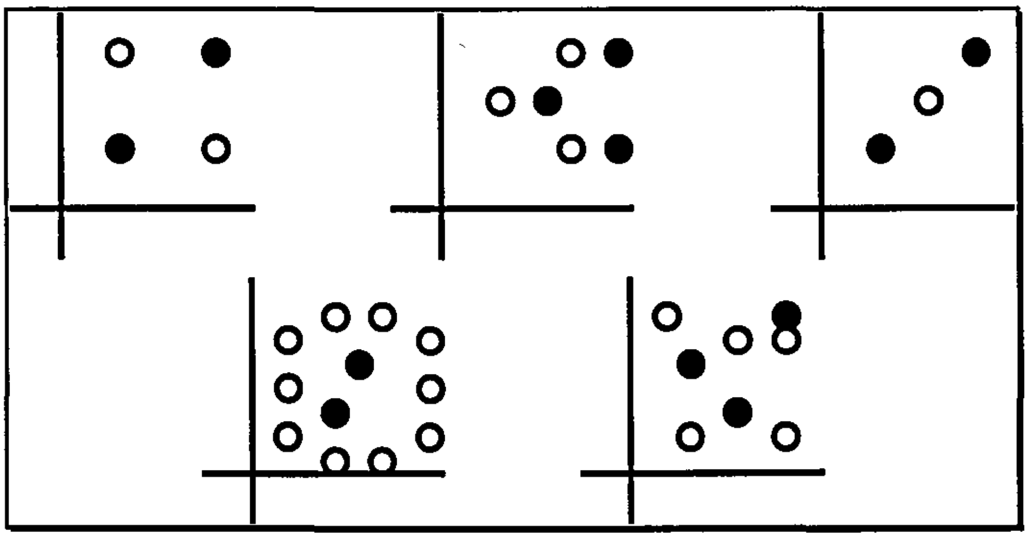


Figure 1.12 Non linearly separable functions

Because a problem is not linearly separable over the inputs given does not mean that a single layer neuron cannot solve it. The problem may be separable over a quadratic of the inputs. To solve this type of problem it is necessary to increase the number of inputs to the neuron. For example the fourth problem in figure 1.12 could be solved using a neuron with inputs of X^2 and Y^2 in addition to X and Y. This produces an oval shaped area as shown in figure 1.13

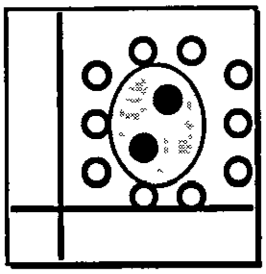


Figure 1.13 A solution using polynomial inputs.

This sort of neuron is known as a Polynomial ADAptive LInear NEuron or a PADALINE. They give considerably more power to a single layer network and solve some of the problems associated with perceptrons. However, it requires the network designer to decide which polynomials should be included in the network structure or to use a heuristic method that adds different polynomial inputs and removes others that offer nothing to the discrimination.

Back Propagation

Work on developing a learning system for multi layer networks faded, but did not die all together. The method of error back propagation for the solution of multi layer problems was first published by Bryson and Ho in 1969. This work was presented in a very formal manner and, perhaps due to its inaccessibility, went unnoticed by the connectionist research community. Due to this lack of publicity, connectionist research was not revived until it was rediscovered, named "back propagation" and properly publicised by Rumelhart, Hinton and Williams [1986 a,b].

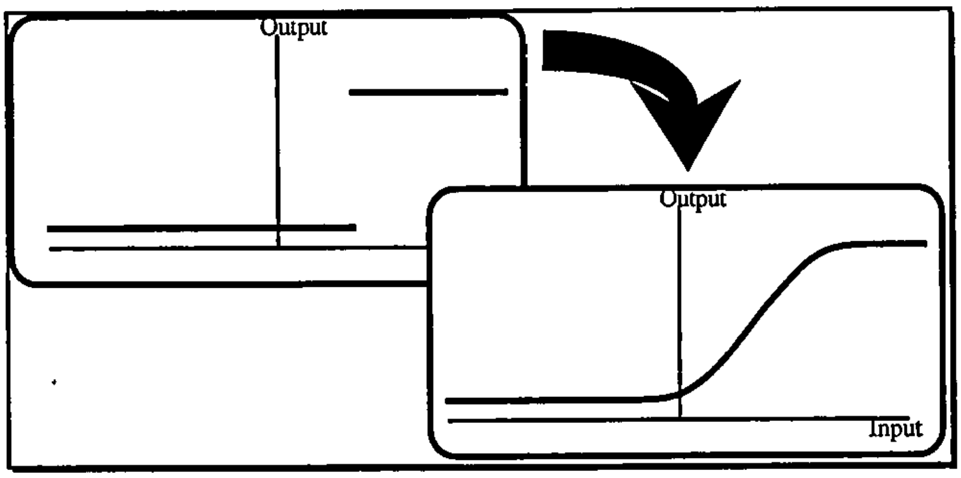


Figure 1.14 The replacement for the threshold function used in back propagation

The idealised neuron used in the perceptron model used a threshold as its activation function. Back propagation calculates an error for each training example. To do this neurons must produce a differentiable continuous output. The threshold of the idealised neuron is normally replaced with the sigmoid function shown figure 1.14.

Consider the function g where g has a parameter P. E.g..

$$g(X) = X + PX^2$$

If we wanted to modify the function g so that it matched some training examples then it would be necessary to calculate a value for P. If an example told us that g(X) should equal Y then it is necessary to find the correct value of P such that

$$g(X) = Y$$

Let the actual result of g(X) = S. Therefore we have an error of E.

$$E = (Y - S) \tag{1.7}$$

The effect of changing P in the value of g(X) can be found by differentiation with respect to P. Let this be called D.

$$D = \frac{d(g(X))}{dP} \tag{1.8}$$

The error can be reduced by adding the value ΔP to P where ß is a small positive number and

$$\begin{aligned} \Delta P &= ß \,.\, E \,.\, D \\ &= ß \,.\, (Y - S) \,.\, \frac{d(g(X))}{dP} \end{aligned} \tag{1.9}$$

<u>Example</u>

$g(X) = X + pX^2$ $\qquad$ $p = 0.1$ $\qquad$ $ß = 0.5$

the correct value of g(1) is 2.1.

<u>Iteration 1</u>

$$\begin{aligned} S &= g(1) & \qquad E &= 2.1 - 1.1 \\ &= 1.1 & &= 1.0 \end{aligned}$$

$$\begin{aligned} D &= \frac{d(g(X))}{dP} & \qquad \Delta P &= 0.5 * 1.0 * 1.0 \\ &= \frac{d(1+P.1)}{dP} & &= 0.5 \\ &= 1 & & \end{aligned}$$

The new value of P after iteration 1 is

$$\begin{aligned} P_{new} &= P_{old} + \Delta P \\ &= 0.1 + 0.5 \\ &= 0.6 \end{aligned}$$

The table below shows several iterations, notice how the error, E, reduces after every iteration.

P	S	E	D	ΔP	P_{new}
0.1	1.1	1.0	1	0.5	0.6
0.6	1.6	0.5	1	0.25	0.85
0.85	1.85	0.25	1	0.125	0.975

This was a very simple example, but it illustrates how a parameter can be modified in order to match a training example. Now consider a neuron in a feed forward neural network.

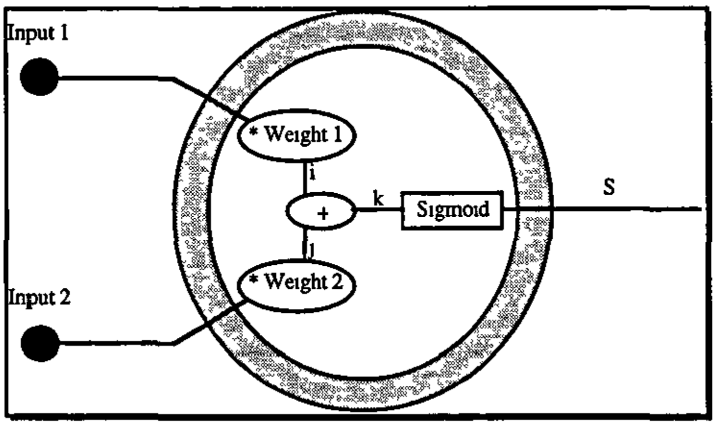


Figure 1.15 The inside of a neuron. The diagram identifies several intermediate values used in the calculations.

Here we have inputs, outputs a transfer function and parameters Weight1 & Weight2. It is the parameters that need to be modified to reduce the errors on the training set, producing a correct neuron. Using equation (1.9) we get

$$\Delta Wt_1 = \beta \,.\, E \,.\, D$$

$$= \beta \,.\, (Y - S) \,.\, \frac{d(g(In_1, In_2))}{dWt_1} \qquad (1.10)$$

where

$$g(In_1, Int_2) = sig(In_1 * Wt_1 + In_2 * Wt_2) \qquad (1.11)$$

therefore

$$\frac{d(g(In_1, In_2))}{dWt_1} = \frac{d(sig(In_1*Wt_1 + In_2*Wt_2))}{dWt_1}$$

$$= \frac{d(In_1*Wt_1 + In_2*Wt_2)}{dWt_1} . \frac{d(sig(In_1*Wt_1 + In_2*Wt_2))}{d(In_1*Wt_1 + In_2*Wt_2)}$$

$$= In_1 . \frac{d(sig(K))}{dK} \qquad (1.12)$$

where K is the product of the inputs and weights (see diagram). Substituting (1.12) into (1.10) gives the equation for calculating ΔWt_1

$$\Delta Wt_1 = ß . (Y - S) . In_1 . \frac{d(sig(K))}{dK} \qquad (1.13)$$

This rule generalises to give the update rule for all the input weights.

$$\Delta Wt_n = ß . (Y - S) . In_n . \frac{d(sig(K))}{dK} \qquad (1.14)$$

or

$$\Delta Wt_n = ß . In_n . \partial_n \qquad (1.15)$$

where

$$\partial_n = E . \frac{d(sig(K))}{dK} \qquad (1.16)$$

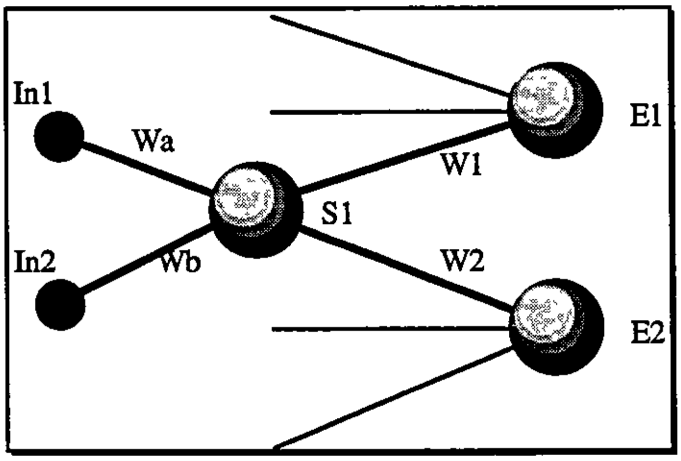


Figure 1.16 Part of a feed forward network

In the network shown the errors of the two right most neurons are known, and the weights W1 and W2 can be changed using the update rule (1.14). However, as yet, we have no way of improving on the input Wa and Wb. Firstly consider just one of the right hand side nodes.

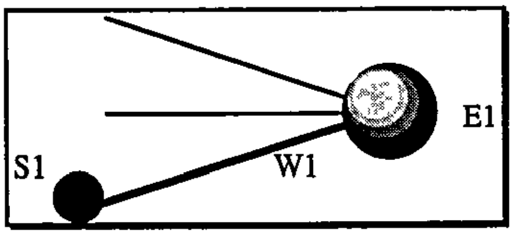


Figure 1.17 A small part of the network shown in figure 1.16

The effect of changing S1 on the output of the neuron is given by

$$\frac{dg}{dS1} = W1 . \frac{d(sig(K))}{dK}$$

Therefore to reduce the error we would make the change ΔS1 to S1

$$\Delta S1 = \beta . E1 . W1 . \frac{d(sig(K))}{dK}$$

However S1 affects the output through more than one route, if we add all the changes together we get

$$\Delta S1 = \sum \beta . E_n . W_n . \frac{d(Sig(K_n))}{dK_n}$$

Substituting in (1.16) gives

$$\Delta S1 = \beta \sum W_n . \partial_n$$

This gives the error associated with S1, the weights Wa and Wb can now be updated using the rule in equation (1.15). Putting all the parts of the algorithm together gives:-

$$W_{new} = W_{old} + \Delta W$$

$$\Delta W = \beta . \text{Input-to-Node} . \partial$$

for output nodes

$$\partial = (Y-S) . \frac{d(sig(K))}{dK}$$

for midnodes

$$\partial = (\sum W_n . \partial_n) \frac{d(sig(K))}{dK}$$

Back Propagation and the Error Surface

Let E_n be a value that represents the error for training example 'n', where y_n is the required output and S_n is the actual output for example n.

$$E_n = {}^1/_2 (Y_n - S_n)^2$$

The total error over the whole training set is E where

$$E = \sum_n E_n$$

If the network gets all the examples exactly correct then the error E is zero, as the difference between the examples and the actual output increases then the size of E increases. The original papers introducing back propagation introduced this error term and derived the update rules from it. They also showed that every iteration reduced the value of E. If E is reduced every iteration then the network must become more correct after every iteration.

Back propagation changes the network by a small amount in each iteration. It can be described as performing gradient descent on a function whose parameters are the weights between neurons and whose result is E.

Problems with Back Propagation

The largest problem associated with back propagation is local minima. Each iteration of the back propagation algorithm reduces the value of E. However that does not mean that there is always a path from the starting point to the global minimum. Consider the graph shown in figure 1.18. If the networks started at points A or C then the correct answer B would be achieved by back propagation. However if E was the starting point then the best achievable answer would be D. This is because there is no mechanism within standard back propagation that can increase the value of E making it impossible to climb from point D to C.

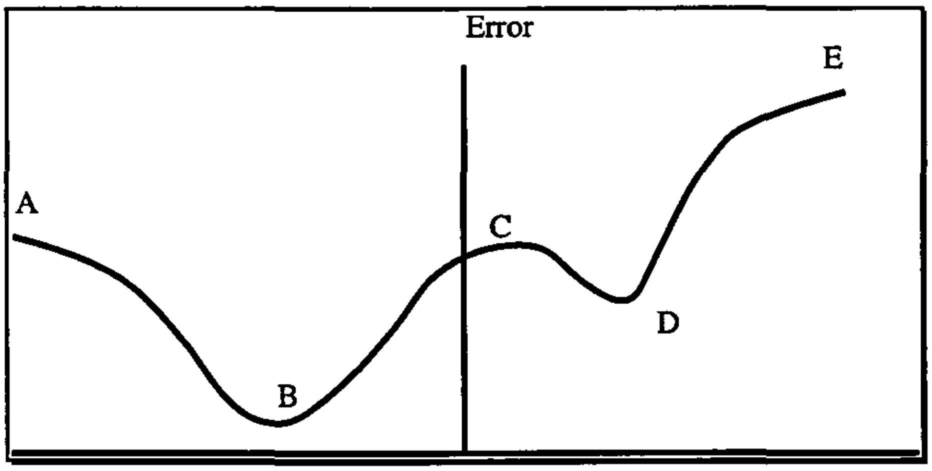


Figure 1.18 An example of an error surface in one dimension.

Similarly a very flat plateau on the error surface will cause the algorithm to converge very slowly. The gradient of the surface is one of the terms in the update rule. If the gradient is very close to zero then the update will be very small.

Enhancement of the learning algorithm

Back propagation has been one of the major research topics since its publication in the mid 1980s. Many extensions and modifications have been considered as the basic algorithm can be exceedingly slow to converge.

Alternative cost functions

The update rules were originally derived from the error function

$$E_n = \frac{1}{2}(Y_n - S_n)^2$$

This is called the quadratic error function, but is not the only possible function. It can be replaced by any function that is minimal when the network correctly classifies the examples and grows as errors are introduced. Solla et al 1988 used the cost function

$$E_n = |Y_n - S_n|$$

Deriving the update rules for back propagation gives the equation

$$\partial = (Y_n - S_n)$$

for the blame assignment function. The gradient term has disappeared. Since the gradient term becomes small when |K| is large, leaving it out accelerates progress in large |K| regions, where the cost surface is relatively flat. On the other hand it gives no acceleration when the cost surface is more sharply curved- when K is near zero.

Fahlman [1989] showed that a compromise was more effective, he derived the update rule

$$\partial = (Y_n - S_n) \cdot \left(\frac{d(\mathrm{sig}(K))}{dK} + 0.1\right)$$

This form restores some of the effect of the differential term, but eliminates the flat spots when |K| is very large.

Finally, we could add parameters to the cost function so as to adjust its steepness during training. It should be possible to smooth out the surface at the expense of detail, and then gradually add in the detail as the network attains the correct region of weight space. One way of doing this is based on the simulated annealing approach [Kirkpatrick et al, 1983]. Here the sigmoid contains the temperature function T. This term has the affect of changing the severity of the sigmoid, as is shown in figure 1.19. The larger the value of T the slower the transformation from zero to one. The new equation is

$$O = \frac{1}{1 + e^{-\frac{X}{T}}}$$

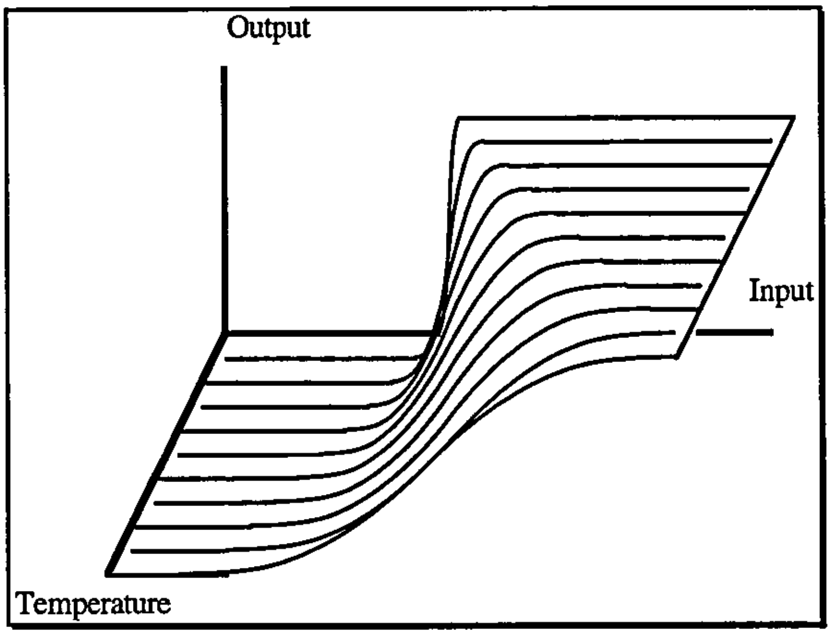


Figure 1.19 The effect of the temperature term T on the sigmoid function.

A high value of T reduces the depth of the minimum in the energy function reducing the likelihood of gettıng stuck in them. By slowly lowering the values of T during the learning the chances of arriving at a correct solution are very greatly improved.

Standard back propagation modifies the weights and the threshold of the neurons. Using the semantıcs developed by J. Horejs et al [1993] this is descnbed by BP[W,V] where

W is the sum of weights

V is the threshold.

in this paper J. Horejs states

"It is clear that by extending the number of parameters and thus the degree of freedom of the network, we can expect a better convergence of the algorithm".

By allowing the temperature to be modified by back propagation in the same way as the weights, BP[W,V,T] a considerable saving in processing was demonstrated. For the X-or problem

BP[W,V]	5000 iterations	0.001 sum squares error
BP[W,V,T]	2000 iterations	0.0006 sum squared error

This type of approach shows a lot of promise, though at the moment there are only empirical results alluding to its success. Work is necessary to produce a theoretical basis for accepting the gains that the extra freedom delivers before it can be relied upon.

Gradient descent is one of the simplest optimisation techniques, but not a very good one. There are many more powerful techniques available, but most are computationally intense and are only suitable for initial off-line training. The simplest approach is the steepest descent line search. Here if we are minimising E(x) we find the direction of steepest descent from the current point, P.

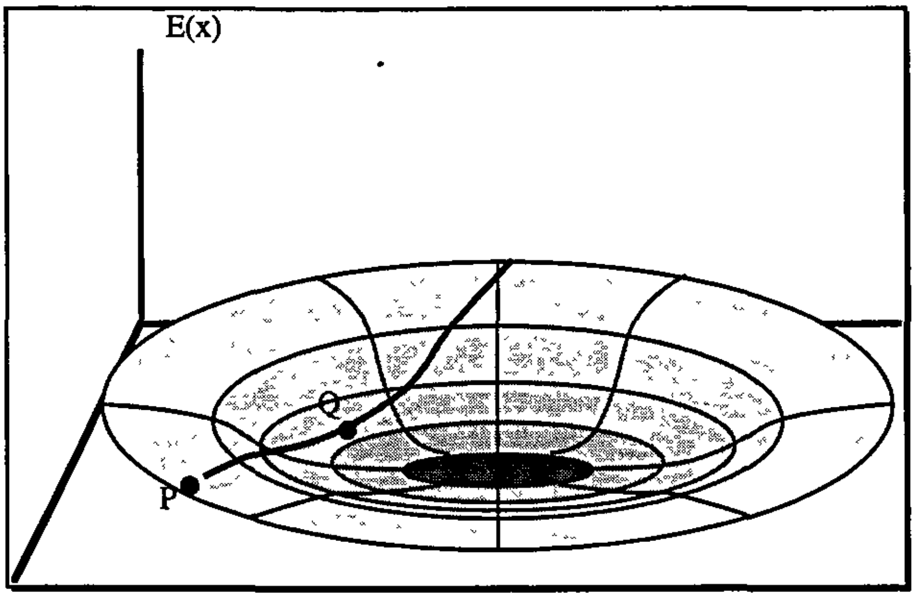


Figure 1.20 A two dimensional error surface. Q is the lowest point on the line of steepest descent from point P.

Searching along this line produces the point Q, which is the minimum value of E(x) on the line. The algorithm is then repeated starting at point Q.

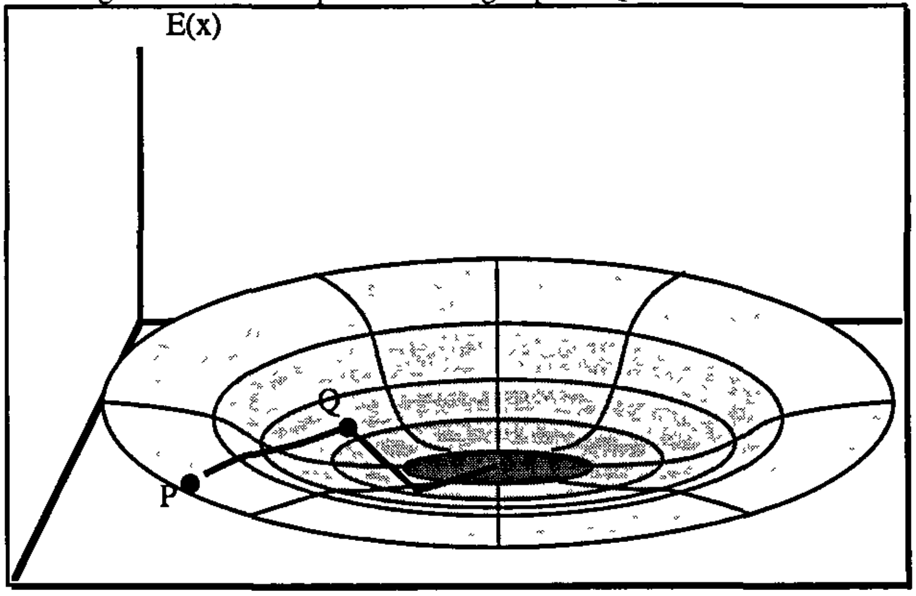


Figure 1.21 The following steps after finding point Q in diagram 1.20.

Several different approaches have been published for finding the one dimensional minimisation of E(x) along the line in a neural network [Luenberger, 1986, Hush and Salas 1988].

The problems with training come from local minima and plateaux on the error surface. All the methods described above have attempted to solve these problems. However, a much simpler approach, adding momentum [Rumelhart and McClelland 1986,Watrous 1987] is commonly and effectively used. The algorithm is to give each weight some inertia, so that it tends to change in the direction of the overall downhill force and not oscillate. On a plateau this means that the weights will accelerate, crossing the flat region much faster. Momentum can also allow back propagation to escape from a shallow local minimum because it is now possible to travel a short distance uphill.

A summary of the feed forward paradigm

Some of the limitations of neural networks, shown by Minsky and Papert in 1969, can only be overcome using hidden layers. However, a multi-layer model of neural networks needs an algorithm which is capable of resolving the blame assignment problem for the hidden neurons. The model, back propagation, successfully uses the differential of the activation function to assign blame, thus producing a gradient descent optimisation. Back propagation has enabled the limitations of the original perceptron model to be overcome and is now a useful general learning algorithm.

Many questions remain unanswered, notably those of speed and reliability. Active research is underway in an attempt to solve these problems by modifying back propagation or changing the optimisation algorithm. Alternative models that modify the structure of the network during the learning stage are also showing some promise [Mezard and Nadal 1990].

Other Neural Models

The models so far described are based on modelling the motor control and perceptive areas of the brain. The biggest area not covered by these models is probably memory. The model by Hopfield [1982] is an attempt to fill this gap.

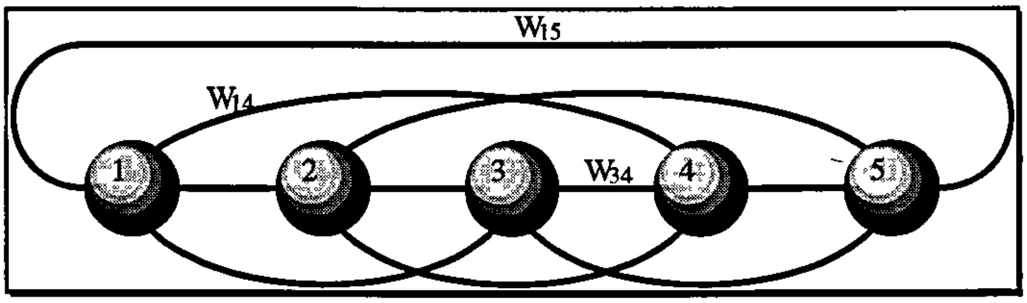


Figure 1.22 A small Hopfield network

Each node in the network is a simple idealised neuron with bi-polar outputs. The network is fully connected, i.e. every node is connected to every other node. Node X is connected to node Y with weight W_{xy}, and $W_{xy} = W_{yx}$. A node fires if the sum of its inputs is greater than its threshold. E.g. Node P_4 will fire if

$$P_3.W_{34} + P_1.W_{14} + P_2.W_{24} > \text{Threshold of } P_4$$

If a node fires its output is set to +1 else it is set to -1. When the model is run a random node is selected, and a test is made to see if, given the values of all the other nodes, the selected node should fire. The output of the selected node is then set to the correct value. Another node is then randomly selected and the process repeated.

There will be states in which every node has a correct value given the state of the other nodes. These states are known as local minima. The aim of the Hopfield model is to design networks in such a way as to represent memories as individual local minima.

Content addressability is possible in a Hopfield network by setting the values of known nodes to their correct value. These nodes are never selected for updating as they are guaranteed correct. By updating the other unknown nodes in the usual way they will find the local minima that corresponds to the known nodes. In this way it is possible to recall a whole picture from a small proportion.

The pictures shown in figure 1.23 have been produced by a 23400 node network, where each node represents a single pixel. The network has been constructed so that there are seven local minima that correspond to seven different images. The images on the right are the minima reached after starting the model with the starting points shown in the left hand column. The model has restored the entire image after being presented with a small proportion of it, this is slow but effective content addressability.

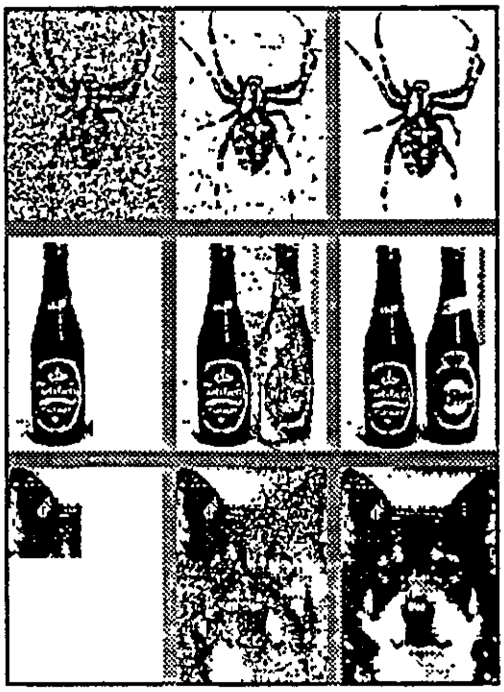


Figure 1.23 Images being recalled after storage in a Hopfield network. The network was constructed to contain 7 different images. By presenting the portion of the image shown in the left column the network restored the rest of the image(right column). The middle column is an intermediate step.

Conceptual Leap and the Hopfield energy function

Conceptual leap is the process of building hitherto non-existent links between data items. The local minima or stored information are local minima on an energy curve. The Hopfield update rule moves around on the energy curve in the same manner that back propagation moved on an energy curve. Similar concepts will be close to one another, for example the two local minima indicated in figure 1.24 are very similar. With conceptual leap it should be possible for the model to link these two concepts.

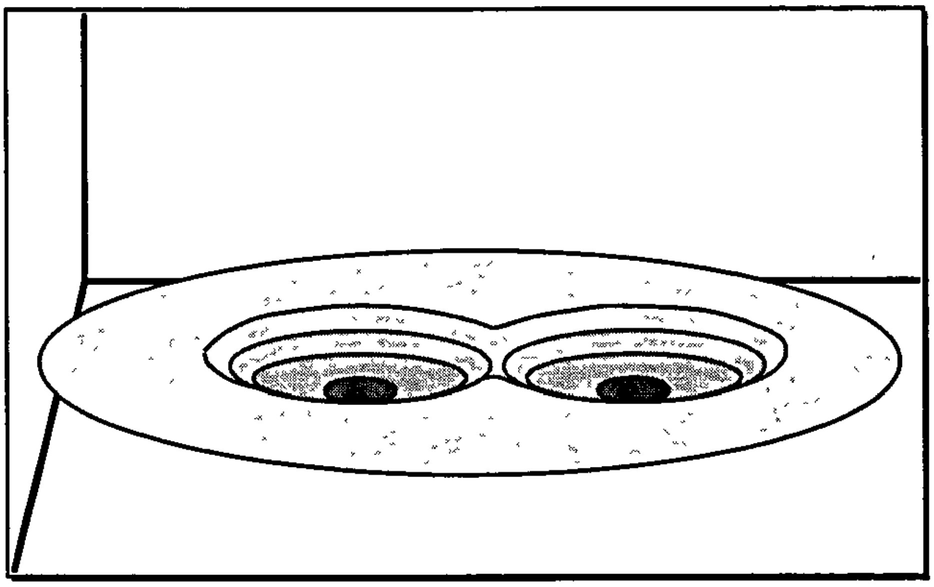


Figure 1.24 Close local minima represent very similar concepts.

The mechanism is very simple. Take a network in a local minima, changing some of its nodes at random, and then let it find a new local minimum. The new minimum will either be the same as the old one or a different but close one. The more random changes that are made the longer the links that are possible and the more tenuous the link between the concepts.

Hopfield originally suggested the Hebb rule [1944] as a training mechanism for the Hopfield model. This can introduce stable states which are not desired. Different solutions have been put forward for solving this problem. Poppel [1987] suggested a trial and error method. The weights are modified in order to widen the basin of attraction of the desired states while reducing that of the undesirable states.

Hopfield [1983] updated his original learning method and introduced the idea of unlearning. The network is randomised, if the network converges to an "unwanted" stable state, then weights are modified so as to unlearn that state. This is the reverse of the original Hebb learning rule.

Another model of significance is due to Teuvo Kohonen [1984]. The model uses a standard sigmoid neuron, but does not use back propagation. Instead it uses a mechanism based loosely on the Hebb rule.

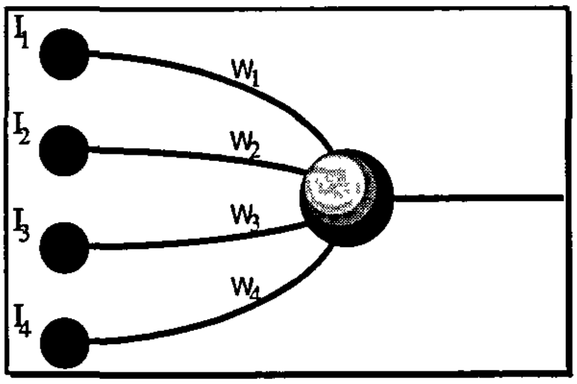


Figure 1.25 Simple four input neuron

The learning mechanism operates as follows; if the neuron receives a signal from connection i and the neuron fires then W_i is reinforced. If the neuron receives no signal from connection i or receives a signal and does not fire then W_i is diminished.

Kohonen was attempting to reproduce the perception systems found in mammals. It covers the reception of the inputs and the processing of the information within the cortex. The main characteristic of these cortices is that neighbouring regions encode similar input signals. Studies have shown that in biological cortices each logical cell encodes a very specific input pattern [Messenger] and there is a lateral interaction dependant on distance [Kohonen, 1984].

Neurons in this model are placed in a two dimensional grid. Each node is connected to every input. When an input pattern is presented the node that gives the highest output value is found. This node is then slightly modified to make its output even higher, making the node represent this input slightly more. If each input is shown many times the neurons will learn to represent all the different types of input. For example, given an ınput space of the English vowel letters, a map lıke that shown in figure 1.26 may be produced.

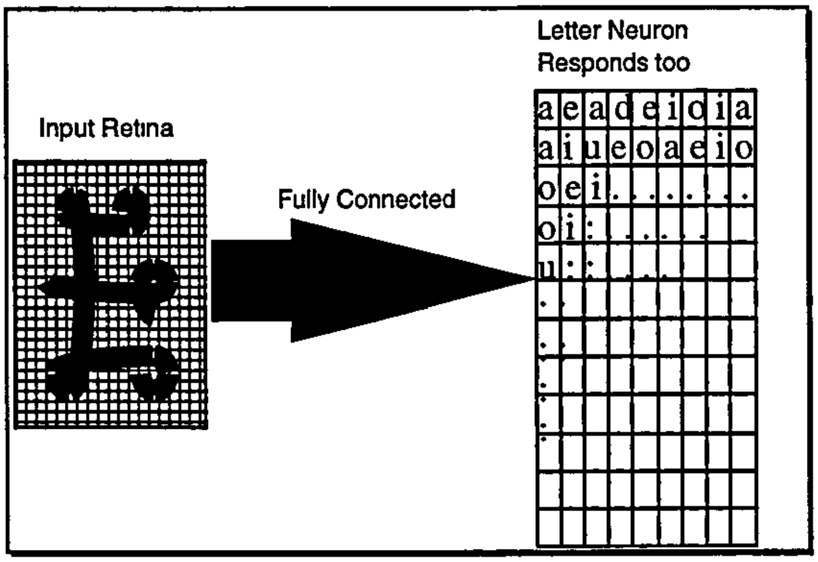


Figure 1.26 The type of map produced when no lateral connection is used in training.

Each neuron has learnt to respond to one individual input, but the model called for similar inputs to be classified by adjacent neurons. This is achieved by training every neuron for every example by a factor, μ. The higher the value of μ the more the neuron is trained to correspond to the input. The best neuron still wants to be trained by the same amount as before so for this neuron μ =1. Neurons that are adjacent to the best neuron need to be traıned slightly less, and neurons that are further away less stıll. The most common function for μ against distance from the best neuron is given in figure 1.27, this is known as the Mexican hat function.

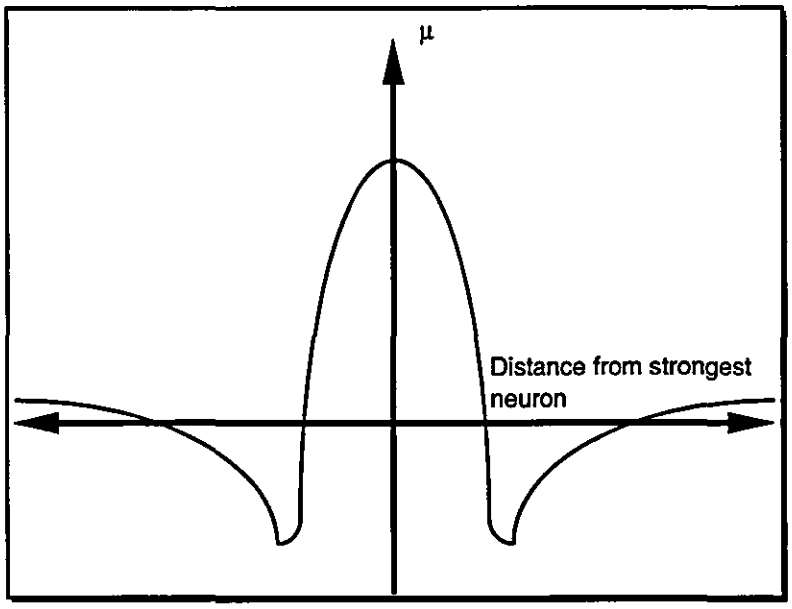


Figure 1.27 The Mexican hat function

This type of lateral influence between the nodes in the map encourages nodes close to each other to respond to similar inputs and those further away to different ones. The resultant map for the English vowels looks something like

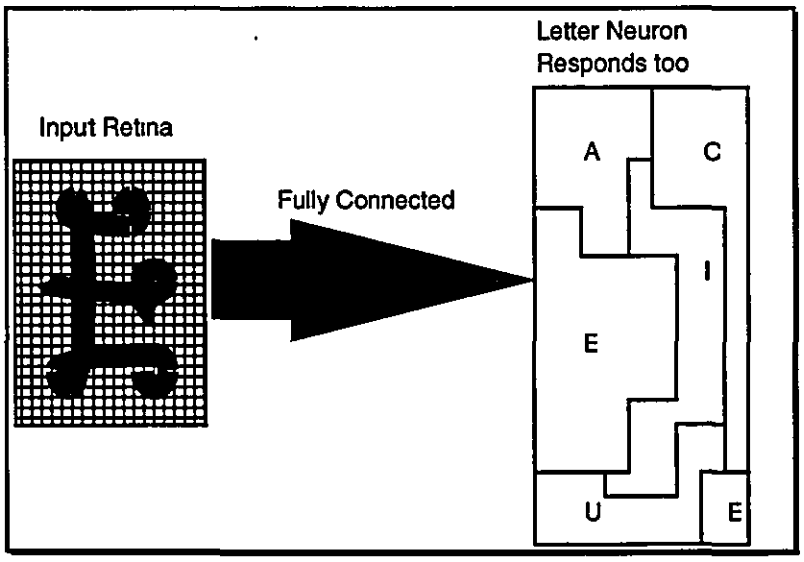


Figure 1.28 The type of map produced when lateral connections are taken into consideration during training.

Now, instead of the neurons responding to each of the letters being distributed across all of the output nodes, neurons corresponding to the same letter have grouped together. This model is most interesting because it is an example of unsupervised learning. The neurons have spread out to cover the whole of the input space. The examples contained no indication of what the outputs should be, directly contrasting with the back propagation and Hopfield models.

Applied research

The vast majority of research work since the advent of back propagation has been in applying the technology to problems and faster implementations of the existing algorithms on parallel machines and dedicated hardware. By comparison, further theoretical development and improvement of the algorithms has been limited to a few small centres. To give an idea of the current uses of neural networks some applications and the papers that present them are listed:

Speech: NETTALK Sejnowski and Rodenberg [1986] developed a network capable of taking written English and producing the phoneme codes required for speech. The success rate was approximately 80% with general English text and was understandable by a human listener.

Adaptive control: Grant and Zhang [1989] developed a network capable of solving the pole balancing problem. This problem is only a toy example but demonstrates all the requirements for developing full control systems for industrial systems.

Forecasting: A share price prediction system. Using data on the fluctuations of a single company the network was able to predict the future changes with a precision of 80%.

There are also many thousands of other uses but it is hoped that the three given show the breath of applications currently under active research.

Engineering Considerations

Even though neural networks are being used in wider and different applications every day there are still some major limitations. Perhaps the largest of these is the amount of trust that can be placed on the hypothesis constructed by a neural network. Messom [1992] produced a systematic method of designing the architecture in an attempt to increase the quality of the engineering involved. This goes some way towards guaranteeing the suitability of the hypothesis as its shape is bound by the network architecture. If the network architecture does not match the needs of the problem then the hypothesis cannot be of the correct shape. The learning algorithm cannot overcome hardware implementation errors. Consider the following problem

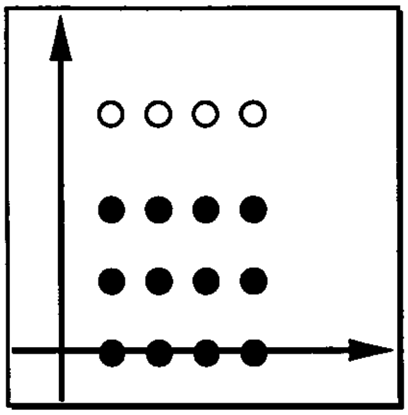


Figure 1.29 The difference between a two and one dimensional implementation.

A node is required to calculate a linear threshold on a single variable input. If a one dimensional implementation is used then this can easily be used, but the two dimensional version requires the weight on the second input to be exactly zero for a correct answer. This is very difficult to achieve. The answer will probably be correct almost all the time but cannot be guaranteed.

There are two differing opinions about how to spread the information about the networks. There are those that favour a localist representation where each node corresponds to a meaningful concept and others that view a distributed paradigm as correct. While I believe that the distributed paradigm is biologically correct, I think that the advantages of the localist representations outweigh the disadvantages.

Distributed representations have a major advantage in robustness. Damage to the system causes a slow degradation of performance as the quantity of damage increases. Localised systems collapse catastrophically when they are damaged. The best example of a distributed system is the brain, which can recover and continue to work after severe injuries.

The major criticism of distributed systems is the unintelligibility of the hypothesis. With a localised paradigm it is sometimes possible is assign meanings to the nodes. However as the networks grow in size beyond several different layers it can be very difficult.

Hinton [1990] expresses the view

> "... the problem is to devise effective ways of representing complex structures in connectionist networks without sacrificing the ability to learn the representations. My own view is that connectionists are still a very long way from solving this problem...".

Effective representation of the network as Hinton asks for will be a major step towards provability of the hypothesis, which in turn I think will allow the connectionist approach to be used in environments hitherto hindered by a lack of trust.

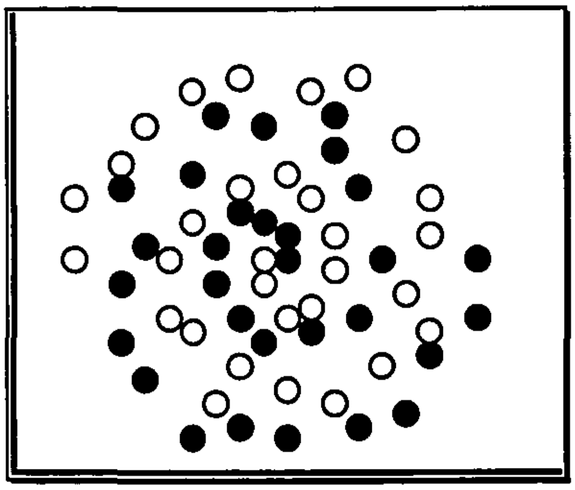


Figure 1.30 The training set used in the two spirals problem

A good example of a problem that does not produce a perfect hypothesis is the spiral problem. The aim is to correctly classify points as being on the black or the white spiral using a network which has only two inputs, the X and Y co-ordinate of the point. The network used to implement this problem is shown below

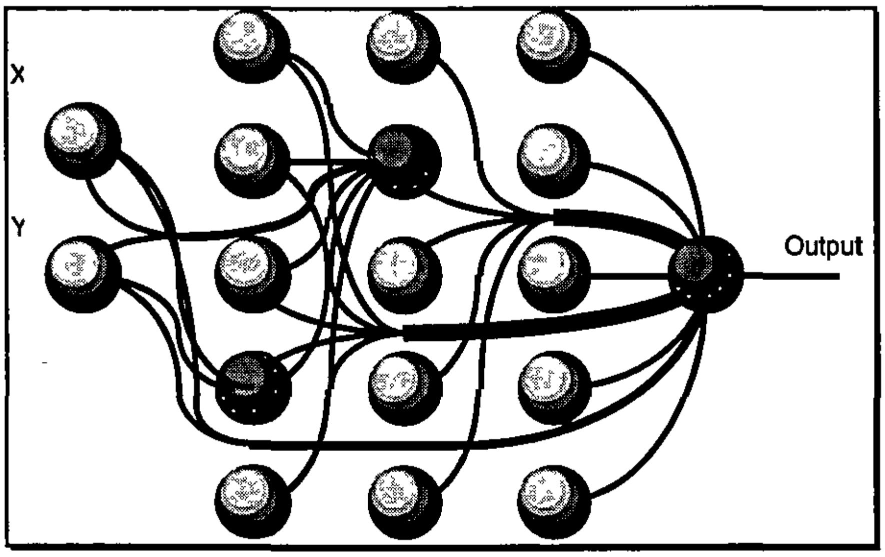


Figure 1.31 The network used in the two spirals problem. Only the connections from the darker shaded nodes are illustrated.

Each node is connected to every node in every subsequent layer. So the X input is connected to every node in layers 1, 2, 3 and the output node. In the diagram only the connections from the darker shaded nodes are drawn. The diagram below shows the output values for each X and Y pair in the input space after the network has been trained. While the network returns the correct value for every training example the network is not an exact representation of the spirals. The hypothesis may be good enough for most applications but must be improved upon for safety critical work.

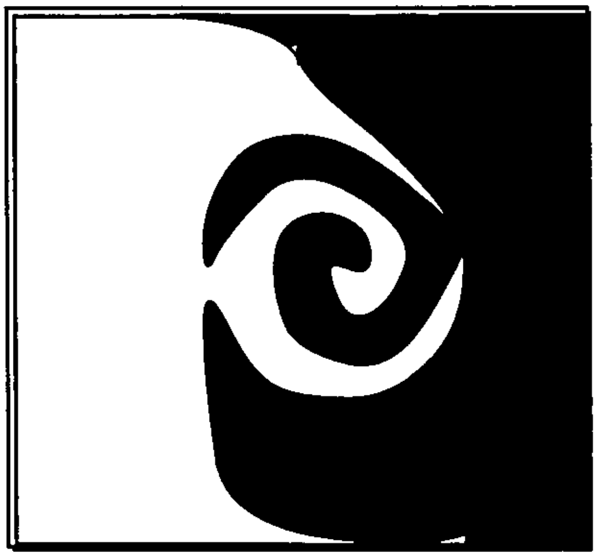


Figure 1.32 The output function of the network used in the two spirals problem.

Summary

The study of neural networks originates from the biological sciences. Artificial neural networks were originally developed as a mathematical model of the activity within the brain. This chapter started by exploring the link between early models and their biological foundations.

The first major step forward came with the simple perceptron model; the simple perceptron model could be trained effectively. This encouraged a huge amount of work by many different research teams leading to several different models of different areas of the brain (the Hopfield and Kohonen models). One of the groups working with the perceptron model were attempting to explore the limitations of perceptrons. Minsky and

Papert showed that a simple model had many limitations and that a multi layer model would be needed for most problems. As no effective training algorithm existed for these models many people turned towards more promising research and the study of artificial neural networks died.

In the mid nineteen eighties the training problem for multi layer models was solved, and widely publicised. The back propagation algorithm once again revived research in this area. More recently the considerations of correct engineering have started to affect the design of the networks. Chapter Two of this thesis considers these problems in more detail, showing the motivation for the research contained in chapters Three Four and Five.

MOTIVATION OF STUDY
CHAPTER 2

Outline Of Chapter

The study that is presented in this thesis is motivated by the lack of reliability that can be placed on a trained neural network. The lack of formal design tools in neural computing has led to the advent of many ad hoc approaches, such as the oversupply and reduction of neurons [Tanii et al. 1989]. These techniques lead to networks that correctly classify their training sets, but no knowledge about their general internal representations has been gained during training.

Neural network training has been well studied, while network reliability and interpretation has received little attention. This chapter examines the elements of a neural network system that are important when verifying its actions over an input domain not in the training set. The differences in verifying correctly engineered and "ad hoc" networks are discussed. Having provided the motivation for the investigations in this thesis an outline of the contents is given, highlighting the new contributions to the field.

Design and Verification

Neural systems have been used in many situations, such as pattern recognition [Rummelhart et al. 1986], fault detection [Govekar et al. 1989, Venkatasubramanian & Chan 1989] and industrial control [Anderson et al. 1990 1991, Jalel at al 1991]. In all these fields is is important that the neural system correctly models the physical properties over the whole operating domain and not just over those elements that appeared in the training set; consider the consequences of producing a network that misclassifies faults on parts that go on to become part of a safety critical device (such as an aeroplane) or a network that acts incorrectly when faced with unusual circumstances as the controller of a chemical plant.

There has been work studying the properties of neural networks [Rummelhart et al. 1986, Minsky et al. 1969, Rosenblatt 1962]. More recently most of the research has switched to reporting the implementations of problems using neural networks [Bailey et al. 1988], but there has been little work presenting rigorous design or verification techniques. A general methodology for both problems is needed before neural systems can be successful and become an accepted and trusted problem solving technique in the industrial, technical and commercial domains.

Some design methods have started to be developed [Messom 1992]. The use of such methods significantly simplifies the internal representation of a neural network. In essence a designed network assigns a simple concept to each neuron, thus moving away from the idea of a distributed hypothesis. Damage to a designed net is catastrophic as opposed to the slow degradation seen in other neural systems. However, given that damage to the network is unlikely in an industrial setting verification would appear to be a more important goal.

Verification and design are strongly interdependent. It is very difficult to verify an ad hoc "hacked" network, and there is little point designing a system if there are no acceptable methods for checking the design. A greater need for verification will help to force the use of a rigorous design methodology in the same way that full software verification has forced the rise of formal methods in software engineering. When designers and customers alike can compare the clean and elegant representation derived from a properly designed network compared to that from an ad hoc system there will be much greater pressure for their use.

Network Structure

Neural systems have a topology. This topology heavily influences the hypothesis that the network can derive. The design systems aim to calculate the correct topology for a given problem. It has been repeatedly stated in this chapter (and shown later) that large networks are harder to analyse. As part of a design analysis and verification system some attention should be given to the size of the network.

Many researchers have looked at network topology [Messom 1992, Tanii et al. 1989]. One area that has not been so popular is the topology within a neuron. Horejs et al. [1993] showed that by changing the sigmoid function dynamically they could alter the learning algorithm. Others have shown that modifying the activation function can produce faster learning. If the topology inside the neurons can alter these characteristics then correct selection of an activation function could reduce the size required in a network.

Thesis Overview

This chapter has argued that there has been a lack of research in the design and verification of neural networks. The previous work that has been undertaken has concentrated on designing Boolean classification networks. It is these networks that are currently most common as control or classification systems.

Chapter three of this thesis looks at how Boolean classification can be analysed. It deals with some limited work presented by Mihalaros [1992] and then goes on to describe a more robust system. A neural network can implement very complex hypotheses. The chapter also looks at how to simplify the ideas extracted from a network, so that they can be used to verify the network or generate more examples to correct the errors.

Boolean classification does not always lend itself to the "Rule extraction" method described in chapter three. Image recognition is one example where a rule has very little meaning. Chapter three closes by considering how these types of problem could be examined.

Whereas chapter three looked at analysing a designed network, chapter four deals with one of the areas not fully explored in the design tools. It considers the internal structure of the neurons that make up a neural network. The aim is to reduce the number of neurons required to correctly model a physical system.

Chapter five also attempts to fill a hole in the current design tools. This chapter looks at how to build a network for modelling Real valued functions, not the simpler Boolean functions. Between the three chapters the analysis of a large class of networks is covered and several holes in the design tools filled.

THE ANALYSIS OF TRANSFER FUNCTIONS

Chapter 3

Outline of Chapter

The chapter starts by explaining the concept of representing the action of a neuron as a form of Boolean function. The chapter proceeds by introducing some of the previous work in this vein, analysing it to show several faults.

Boolean functions are adopted as a representational framework, and a fast algorithm is presented for deriving the transfer function. These functions are then used to convert a neural network into a rule based system. The rule bases are then simplified using techniques similar to noise reduction in image enhancement to extract information about the internal hypothesis of the original network.

The limitations of rule induction from networks are identified and then demonstrated. An approximate solution is shown for analysing networks that are too big for precise analysis.

Boolean representation of a neuron

Neural networks are able to accept continuous valued inputs. In the idealised neuron model, thresholding means that all transformations beyond the first are Boolean transformations, whereas the first layer may be regarded as a quantisation layer. The quantisation layer splits the input space into regions which are then combined logically by the subsequent layers. In attempting to analyse neural networks only the transformations of the second and subsequent layers are addressed; the first layer may be interpreted as the results of relational operators comparing real valued inputs. The first layer results can be stored as a set of inequalities which can be substituted into the analysis of the later layers.

The use of a sigmoid function in later, more powerful models changes the view of thresholding somewhat. However, with a fully trained network the sigmoid function will behave as a very close approximation to the threshold function. There are two reasons for the close approximation to the threshold function :-

1) As the networks hardens the weights grow in size. This makes the range of inputs for which the output is not close to Boolean very small.

2) If simulated annealing is used to speed up training then as the temperature approaches zero the activation curve becomes very close to a threshold.

Booleans through truth tables

It is possible to produce a correct Boolean representation by applying every possible pattern to the inputs of the network, calculate the matching output value for each one producing a truth table. This truth table can then be turned into a Boolean function using an algorithmic implementation of Karnaugh maps [Dowsing et al, 1986]. Both stages of this method have a time complexity of $0(2^n)$, giving a total time complexity of 2^{n+1}. The time complexity is prohibitively large, preventing this approach being used on any reasonably sized network.

O and A operators

Although a threshold neuron can always be represented as a Boolean function, the form of this function cannot be restricted to a representation where each input variable is mentioned only once. The interest in such representations stems from scalability, as the search space for the correct Boolean function grows exponentially as the number of inputs rises. If a set of operators can be found that represents any neuron while mentioning each operand only once, then an effective algorithm could be produced for finding a concise representation.

Previous investigations into this problem used a piece meal approach, splitting off the Boolean problem space into a class of operators which are referred to as the O and A operators as they are generalisations of OR and AND. This produced interesting results on small test data sets in two dimensions, but work presented in this chapter shows they fail to generalise properly to problems in higher dimensions.

The problem space covered by the operators was shown to be complete in up to three dimensions by Mihalaros [1992] and therefore to provide a means of representing all possible neurons with an input dimension of three or less. This work by Mihalaros presented no complete analysis of how to find the O-Operator that matched a general neuron, or even if all possible neurons in higher dimensions were covered. Perhaps more significantly, and what encouraged further work, is that the traditionally hard problem of n-input parity neural networks can be simply represented using O-Operators [Messom 1992].

Structure of O and A operators

An O-Operator is a Boolean function (therefore it can only evaluate to True or False) and is written

$$O^n_m(I_1, \ldots, I_m)$$

where the evaluation is TRUE if at least n of the m inputs are true. For example a three input "OR" function $(I_1 \vee I_2 \vee I_3)$ could be represented as

$$O^1_3(I_1, I_2, I_3)$$

In another example the O-Operator

$$O^2_3(I_1, I_2, I_3)$$

is one representation of the Boolean function

$$(I_1 \wedge I_2) \vee (I_1 \wedge I_3) \vee (I_2 \wedge I_3)$$

The A operators are effectively the complementary function based on AND,

$$A^n_m(I_1, \ldots, I_m)$$

The evaluation is true if at least 1 input is TRUE from every combination of n inputs, for example

$$A^1_3(I_1, I_2, I_3)$$

is a representation of the Boolean function

$$(I_1 \wedge I_2 \wedge I_3)$$

and

$$A^2_3(I_1, I_2, I_3)$$

is one representation of the Boolean function

$$(I_1 \vee I_2) \wedge (I_1 \vee I_3) \wedge (I_2 \vee I_3)$$

Matching a general neuron to an O operator

All the calculations shown in this chapter assume that the neuron has outputs in the range 0 to 1 as opposed to -1 to 1. The choice of 0..1 neurons simplifies many of the arguments and proofs. As 0..1 and -1..1 neurons are isomorphic the proofs are equally valid for -1..1. The possible functions that can be represented by a single neuron of this type fall into distinct groups. Each of these groups will contain exactly one natural neuron and possibly some real ones. Where we define Natural and Real to be :-

Natural: All the weights and the bias of the neuron are positive, i.e.
The activation function is $W_1*Input_1 + W_2*Input_2 + ... +W_n*Input_n > B$ and
$W_1 \geq 0, W_2 \geq 0, ..., W_n \geq 0, B \geq 0$.

Real: At least one of the weights or the bias of the neuron is negative, i e.
The activation function is $W_1*Input_1 + W_2*Input_2 + ... +W_n*Input_n > B$ and
$W_1 < 0$ or $W_2 < 0$ or ...or $W_n < 0$ or $B < 0$.

The O-Operator representations of all the neurons in a group are similar, the only difference being that for a real neuron some of the input variables will be negated. All the O-Operators below represent neurons in the same group, the first is the natural neuron, subsequent O-Operators represent some of the real neurons.

$O^2_5(A, B, C, E, U)$	-	The Natural Neuron
$O^2_5(A, \neg B, C, E, U)$	-	A Real Neuron
$O^2_5(A, B, \neg C, \neg E, U)$	-	A Real Neuron
$O^2_5(\neg A, \neg B, \neg C, \neg E, \neg U)$	-	A Real Neuron

One way to match a real neuron to an O-Operator is to convert the real neuron into a similar natural neuron. Matching this new problem to an O-Operator gives a solution that can be modified to represent the original problem by negating some of the inputs. This means that the target of matching a neuron to an O-Operator has been reduced to two separate problems:-

1) How to find the solution for a real neuron by solving a similar natural one.
2) How to find an O-Operator that matches with a natural neuron.

Finding the solution for a real neuron by solving a similar natural one

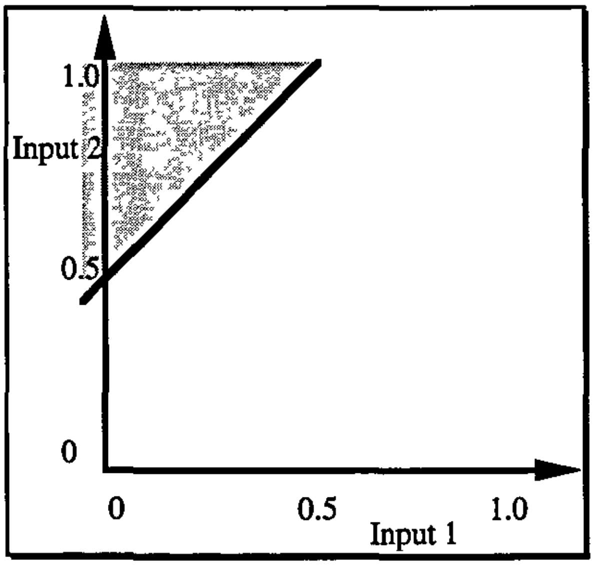


Figure 3.1. A two dimensional neuron with one negative and one positive weighted input, represented as a hyperplane on the graph.

The shaded region in figure 3.1 is represented by the inequation

$$\text{Input 2} - \text{Input 1} \geq 0.5$$

To convert this representation of a real neuron into a natural form it is necessary to remove the negative weight without affecting its Boolean characteristics. The negative weight associated with Input 1 can be modified by moving the origin of the axis to position (1,0) and then reflecting them. The equation for the graph using the new axes is

$$\text{Input 2} + \text{Input 3} \geq 1.5$$

where

$$\text{Input 3} = \neg\, \text{Input 1}$$

This gives a new diagram shown in figure 3.2. If an O-Operator, F, is found for this new neuron then all instances of Input 3 in F could be replaced by ¬Input 1 to give a correct solution for the original problem

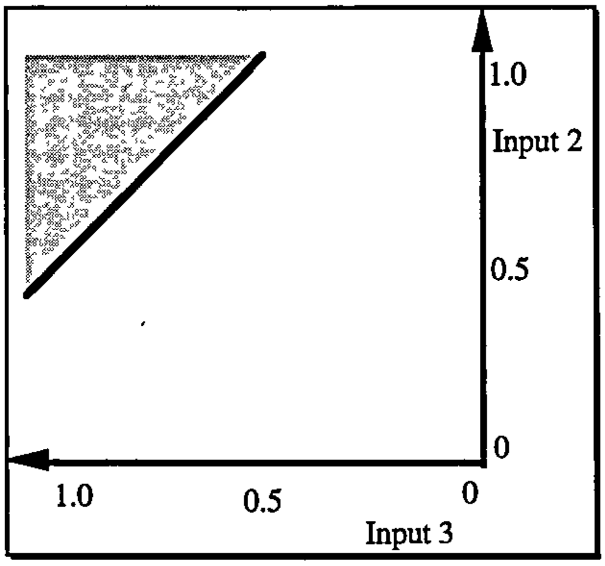


Figure 3.2. The transformed hyperplane with all positive weights.

Any inequality can be converted in this way. In general let the original inequality be

$$w_1I_1 + w_2I_2 + \ldots.. + w_dI_d > B$$

Let the new inequality be

$$W_1I_1 + W_2I_2 + \ldots.. + W_dI_d > D$$

then

$$W_n = |w_n|$$

$$D = B - \sum f(w_n)$$

Where

$$f(X) = \begin{cases} 0 \text{ if } X \geq 0 \\ X \text{ if } X < 0 \end{cases}$$

How to find an O-operator that correctly represents a natural neuron

The function represented by the natural neuron can be illustrated on a layered graph that represents the possible Boolean input space. The nodes of the graph are connected if they differ by only one input and are on the same layer if they have the same number of inputs set to true (figure 3.4). The four dimensional function shown in figure 3.3 would be represented as the graph shown in figure 3.4.

Inputs	I1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	I2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	I3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	I4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Outputs		0	0	0	1	0	0	0	1	0	0	0	1	0	1	1	1

Figure 3.3. An arbitrary four dimensional linearly separable function.

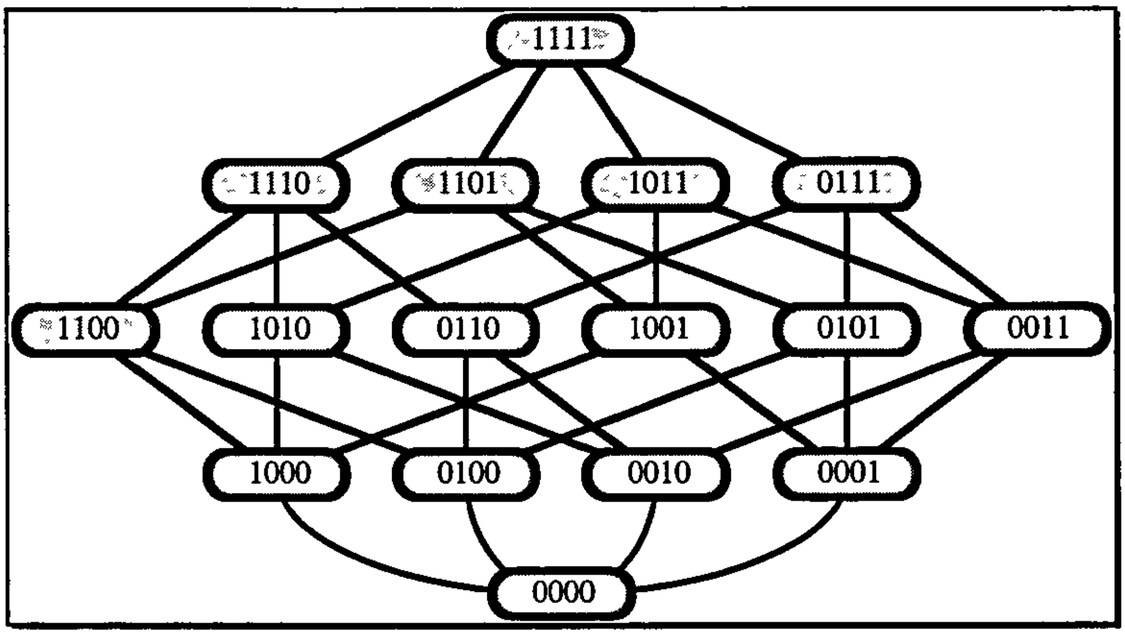


Figure 3.4. The four dimensional function tabulated in figure 3.3 shown as a graph.

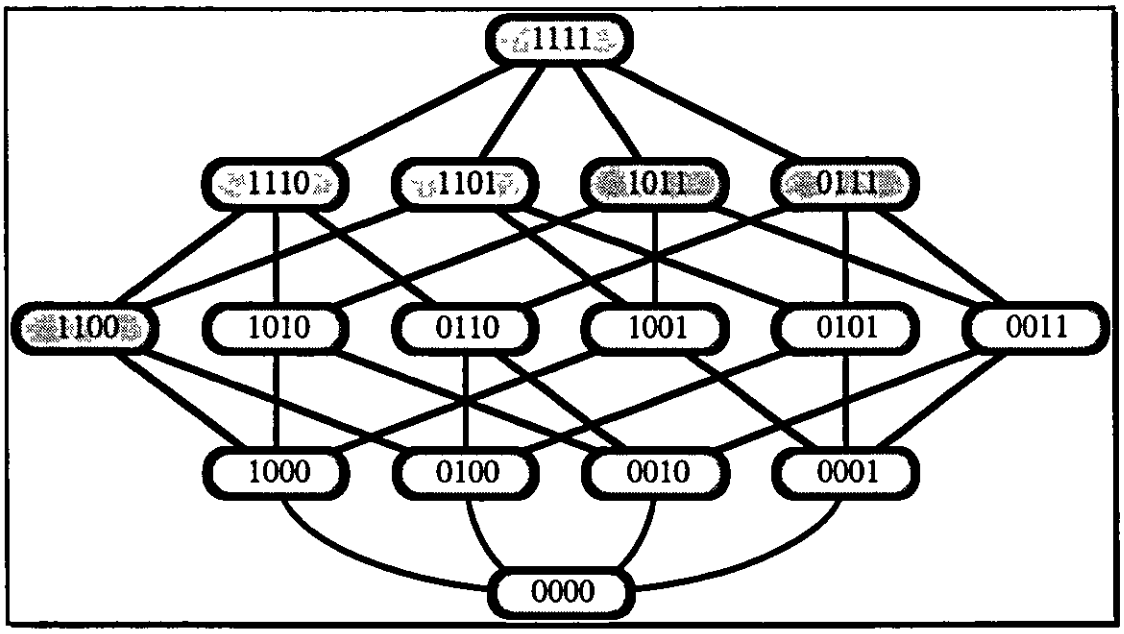


Figure 3.5. The darker shaded nodes represent the infimums of the shaded nodes in figure 3.4.

Only graph nodes that have no children need to be represented in the O-operator; we are only interested in infimums. The threshold is an "at least function" and the infimums are the "least" elements. Finding the O-Operator for these nodes automatically covers the others. Figure 3.5 shows the set of infimums taken from the lattice shown in figure 3.4.

Each of the infimums on the graph can be represented by an O-Operator

$$O^{n}_{n}(I_1 \ldots I_n)$$

where $I_1 \ldots I_n$ are the inputs that are set to one in the required node. e.g. if the required infimum is (1,0,1,1) then the O-Operator is

$$O^{3}_{3}(I_1, I_3, I_4)$$

Where there is more than one infimum any one of them is sufficient to cause a TRUE evaluation. If there are Z infimums than the O-Operator

$$O^{1}_{Z}(O^{n1}_{n1}(A_1 \ldots A_{n1}), O^{n2}_{n2}(B_1 \ldots B_{n2}), \ldots\ldots, O^{nZ}_{nZ}(K_1 \ldots K_{nZ}))$$

correctly represents this property. This O-Operator needs to be simplified in such a way as to remove multiple references to the same input. Some of the transformation rules given by Mihalaros [1992] can be used to achieve this.

1) $O^{1}_{1}(X) \quad \equiv \quad X$

2) $O^1{}_d(O^n{}_n(I_{a1}, \ldots\ldots I_{b1}) \ldots\ldots O^n{}_n(I_{ad} \ldots\ldots I_{bd})) \quad \equiv \quad O^n{}_m(I)$

where $I = \{I_1 .. I_m\}$ and all possible sets of n elements from I were the operands for one of the original O-operators.

This rule represents the main power of O-Operators, the ability to represent some complex Boolean functions very simply.

3) $O^1{}_d(O^{n1}{}_{n1}(I_1) .. O^{nd}{}_{nd}(I_d)) \equiv O^k{}_k(x_1,x_2, \ldots ,x_{k-1},O^1{}_d(O^{r1}{}_{r1}(J_1) .. O^{rd}{}_{rd}(J_d)))$

where $I_1 \ldots\ldots I_d$ are sets of operands,

$I_1 \cap I_2 \cap I_3 \cap \ldots \cap I_d = \{x_1,x_2, \ldots ,x_{k-1}\}$

$J_w = I_w - \{x_1,x_2, \ldots ,x_{k-1}\}$

$r_w = n_w - |\{x_1,x_2, \ldots ,x_{k-1}\}|$

Rule 3 removes common operands from the infimums. If a set of operands appears in every infimum then they can be simplified so that they only appear once in the equation.

4) $O^{1}_{d}(O^{n1}_{n1}(I_1) .. O^{nd}_{nd}(I_d)) \equiv$

$O^{1}_{2}(O^{1}_{a}(O^{n1}_{n1}(I_1) .. O^{na}_{na}(I_a)), O1b(O^{n(a+1)}_{n(a+1)}(I_{(a+1)}) .. O^{na}_{na}(I_d)))$

where $I_1....I_d$ are sets of operands,

$d = a+b$

This rule is splitting the problem into two smaller problems. As the aim of the simplification is to remove multiple references to input variables this rule should not be used if

$$(I_1 \cup I_2 \cup I_3 \cup ... \cup I_a) \cap (I_{(a+1)} \cup I_{(a+2)} \cup ... \cup I_d) \neq \emptyset$$

Doing so in this case causes references to the same variable in both sub problems, where there is no way of recombining them.

The other transformation rules presented by Mihalaros[1992] either introduce or rely on a combination of an input and its negation. In the representation of a neuron it is not possible for a variable and its negation to appear in different infimums, we also do not want to introduce a variable's negation as we are then using two references. All the other rules are not suitable for use when trying to remove multiple references. If none of the first four rules can be used on the current O-Operator then no further simplification is possible.

Example 1

Calculate the O-Operator for the four dimensional function with the infimum nodes (1,1,0,0) and (1,0,1,1). The O-Operators calculated for the two infimums are

$$O^2_2(I_1, I_2) \ \& \ O^3_3(I_1, I_3, I_4)$$

making the starting point for simplification

$$O^1_2(O^2_2(I_1, I_2), O^3_3(I_1, I_3, I_4))$$

Step 1 Applying rule 3 gives

$$O^2_2(I_1, O^1_2(O^1_1(I_2), O^2_2(I_3, I_4)))$$

Step 2 Apply rule 1. There is now no more simplification necessary, the final answer is

$$O^2_2(I_1, O^1_2(I_2, O^2_2(I_3, I_4)))$$

Example 2

Calculate the O-Operator for the four dimensional function shown in figure 3.6.

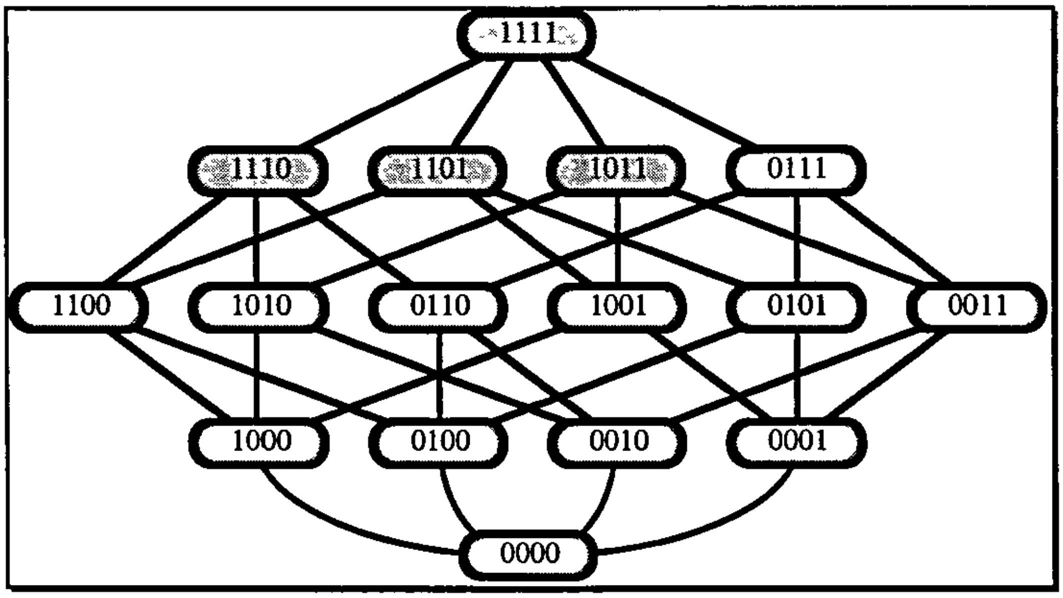


Figure 3.6. A four dimensional function.

The three infimums are (1,1,1,0) , (1,1,0,1) , (1,0,1,1). Therefore the starting O-Operator is

$$O^1_3(O^3_3(I_1, I_2, I_3),\ O^3_3(I_1,I_2,I_4),\ O^3_3(I_1,I_3,I_4))$$

Step 1 Applying rule 3 gives an answer of

$$O^2_2(I_1, O^1_3(O^2_2(I_2, I_3),\ O^2_2(I_2, I_4),\ O^2_2(I_3, I_4)))$$

Step 2 Applying rule 2 gives an answer of

$$O^2_2(I_1, O^2_3(I_2, I_3, I_4))$$

Counter example of O-operator completeness

For the function represented by the graph

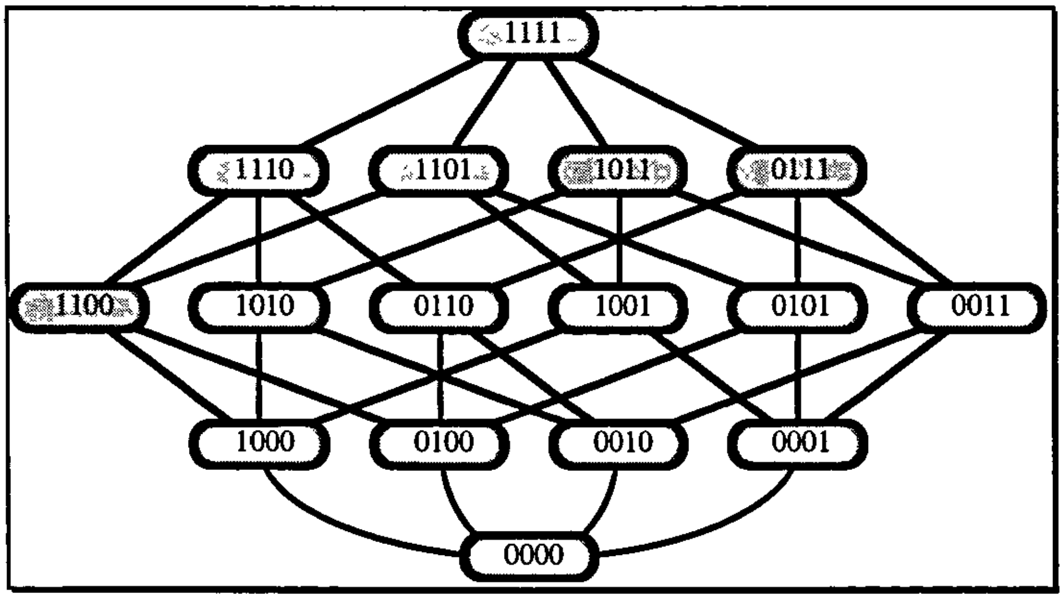


Figure 3.7. A four dimensional function that cannot be represent by O-Operators.

The infimums are (1,1,0,0) , (1,0,1,1) and (0,1,1,1). Therefore the initial O-Operator is

$$O^1_3(O^2_2(I_1, I_2),\ O^3_3(I_1, I_3, I_4),\ O^3_3(I_2,\ I_3,\ I_4))$$

None of the first four rules can be applied for the following reasons:

Rule 1:	There is more than one O-Operator in the group.
Rule 2:	Not all the values of n are the same.
Rule 3:	No operand is common to every member of the group.
Rule 4:	The group cannot be split as every member would be in the same subgroup.

Therefore the function cannot be represented without using a negated version of one of the inputs, breaking the restriction of multiple representation. An exhaustive search of all possible 4 input O-Operators provides an alternative verification. As the function is linearly separable it can be represented using a neuron. Therefore O-operators are incomplete. Unfortunately the incomplete elements of the O-operator are not representable using A-Operators, as O and A operators are only transformations of each other [Mihalaros, 1992].

It has been shown that O-Operators are not complete for four or more dimensions and cannot, in general, be used to analyse networks in higher dimensions. Where it is possible to use O-Operators they provide a concise and clear representation and are ideal for large networks with a low connectivity. Unfortunately, as the number of dimensions increases the proportion covered by O and A operators falls until they are virtually useless.

Boolean functions as the representation

O and A operators are incomplete, they do not have the flexibility to represent all possible neurons. The aim of finding some form of representation that mentions each input once is, in the authors opinion, basically flawed. A more general technique must be adopted. The best understood technique for representing logical functions is traditional Boolean Logic. Although the representation for a complex neuron can grow unacceptably long, Boolean logic can still provide a powerful tool. The aim of the next section of work is to produce a system for analysing neurons in terms of a Boolean rule.

Converting natural neurons to Boolean functions

Natural neurons can be turned into Boolean functions with relatively little effort. It is necessary to find which sets of inputs are the minimum required to fire the neuron (these are the same infimums that it was necessary to model when using O-Operators). This problem is similar to the classic knapsack problem which has an exponential time complexity .

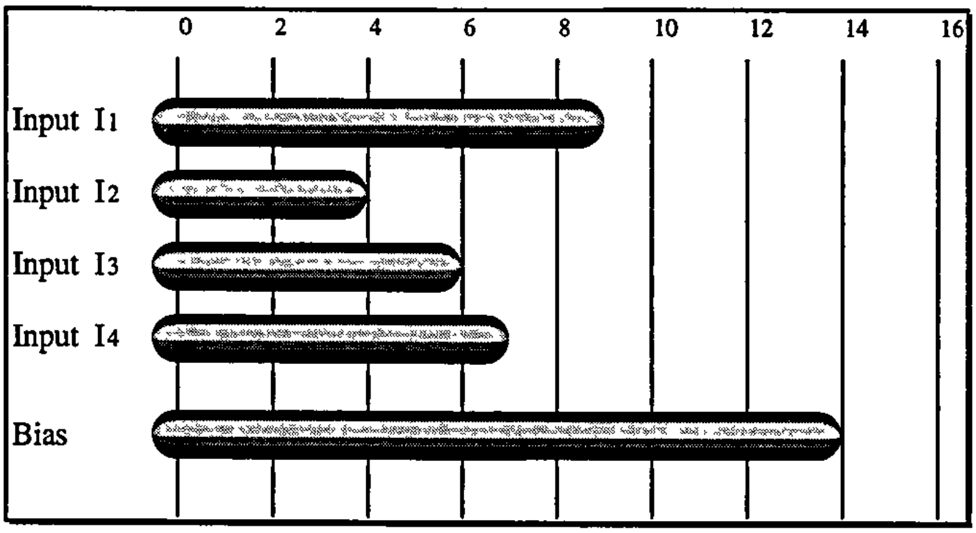


Figure 3.8. Representing a natural neuron as a knapsack problem where the input "parcels" are fitted into the bias "package".

Instead of trying to find the set of inputs that exactly fits the bias, the required answer is all the sets of inputs that just exceed the bias. For example in figure 3.8 (I_1 & I_4) just exceeds the bias, as 9+7 > 14, but if either of the inputs were to be removed then the total falls below the required threshold, (I_1 & I_3 & I_4) does not just exceed the bias as removing I_3 would not make the total fall below 14. The complete set of infimums for the problem shown in figure 3.8 is

I_1 & I_3	$9 + 6 > 14$.
I_1 & I_4	$9 + 7 > 14$.
I_2 & I_3 & I_4	$4 + 6 + 7 > 14$.

This represents the required Boolean function and can be transformed easily to the conventional Boolean format:

$$(I_1 \wedge I_3) \vee (I_1 \wedge I_4) \vee (I_2 \wedge I_3 \wedge I_4)$$

An algorithm to produce one part of the result is given as a pseudo code procedure below. By backtracking the procedure it is possible to derive each part in turn until all the infimums have been produced.

```
knapsack(Array_of_Inputs, Bias, Answer)
     If Bias ≥ 0 then
          for each element, X, in Array_of_Inputs
               New_Array = All elements after X in Array_of_Inputs
               knapsack(New_Array, (Bias - X), Sub_Answer)
               Answer = append([X], Sub_Answer)
     else
          Answer = []
  end
```

Complexity analysis of proposed system

Let C(X) be a measure of complexity proportional to the time taken to calculate the answers. To produce an answer when there is one weight requires one call to the algorithm.

$$C(1)= 1$$

To produce an answer when there are two weights requires two calls to the algorithm.

$$C(2)=2$$

To produce an answer when there are N weights requires P calls to the algorithm.

$$C(N) = P$$

where P is the initial procedural call plus the sum of the recursive calls made to solve each sub problem. The algorithm produces N-1 subproblems for a problem with N inputs. The subproblems are smaller knapsack problems over the last X original inputs for X = (N-1), (N-2), .., 2, 1.

$$C(N) = 1 + \sum_{1 \quad N-1}^{A} C(A)$$

$$C(N) = 2^{N-1}$$

The worst case time complexity for the solution of a natural neuron is $O(2^{N-1})$. This is still exponential, but unlike the exhaustive search method described earlier this can be reduced further, or left with a complexity saving of approximately four times compared to the $O(2^{N+1})$ of exhaustive search.

Methods of reducing the complexity

The algorithm will only start to approach the maximum complexity when most of the weights are required to make the neuron fire, i.e. the decision hyperplane is a long way from the origin. This means an improvement can be made on solution time by moving it closer. If the hyperplane is closer to the opposite corner of the unit hypercube than the origin then viewing the problem from the opposite corner and solving the opposite problem is more efficient. Reflecting the axis along the line y = 1-x will bring the hyperplane less than half way from the new origin. The procedure will then need less recursive loops to find its answers.

e.g.

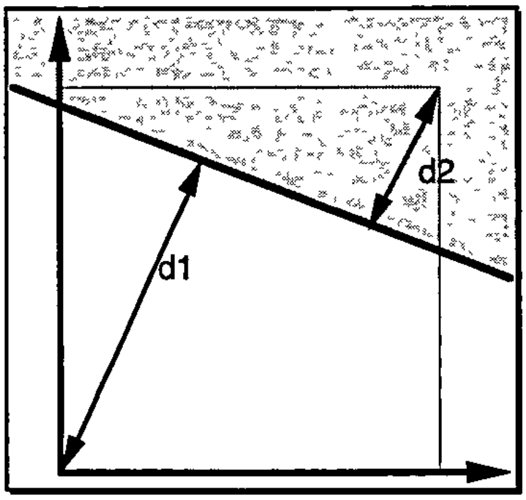


Figure 3.9. Example of a hyperplane implemented by a neuron requiring reflection.

The neuron in figure 3.9, let us call it P, represents a hyperplane that is more than halfway from the origin (d1 > d2). Let G be second neuron such that

$$G = \neg P$$

and then rotate the axis so that G is Natural, figure 3.10.

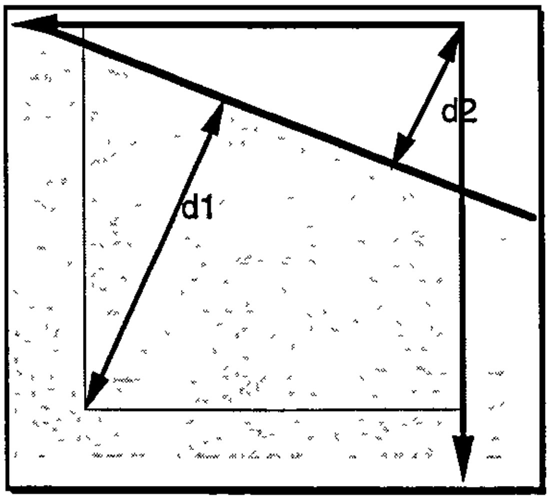


Figure 3.10. the hyperplane shown in figure 3.9 after reflection.

We know d1> d2 therefore the hyperplane is less than halfway to the opposite corner. Once the Boolean function for G, B(G), has been calculated the Boolean function for P is $\neg B(G)$. Using this method it is possible to find the solution for long problems in a vastly reduced time.

When to solve the opposite problem

It has been stated that the problem should be reflected if the hyperplane is more than halfway to the opposite corner of the unit hyper cube. The problem of deciding what constitutes being more than half way from the origin is not simple. The obvious, and best method, would be to decide if more than half of the volume of the hypercube is cut off from the origin by the hyperplane. This can seem a very complex task, but analysis shows that if the boundary is flat, which it must be as it is a single hyperplane, then the side of the hyperplane with most volume is the side that contains the point exactly in the centre of the cube. Therefore to test if reflection is necessary it is only necessary test to see if the point halfway from the origin is above or below the hyperplane. If it is above then no reflection is necessary, otherwise reflect. To perform this test, reflect if

$$\frac{\sum \text{Weights}}{2} < \text{Bias}$$

Forcing the derived expression into a minimal representation

The algorithm so far described will return a correct solution. For the purposes of a useful system the returned answer must not only be correct but also minimal. This can be achieved by sorting the weights into decreasing size before entering them. A further advantage of this is that sorting is a good heuristic for reducing the average processing time to find the solution (figure 3.11). Let the input to the algorithm be weights

$$W_1,\ W_2,\ W_3, \dots W_n$$

and the bias value B. The mistakes that cause the answer to be non minimal occur when

$$W_1 + W_2 + W_3 + \dots + W_m > B \qquad [3.1]$$

$$W_2 + W_3 + \dots + W_m > B \qquad [3.2]$$

Because the algorithm stops as soon as the bias is achieved

$$W_1 + W_2 + W_3 + \dots + W_m - W_m < B \qquad [3.3]$$

If the weights were in sorted order then

$$W_1 > W_m \qquad [3.4]$$

Substituting [3.4] into [3.3] gives

$$W_1 + W_2 + W_3 + \dots + W_m - W_1 < B$$

$$W_2 + W_3 + \dots + W_m < B \qquad [3.5]$$

which contradicts [3.2], therefore, if the weights are sorted, the two requirements for added complication in the answer cannot occur.

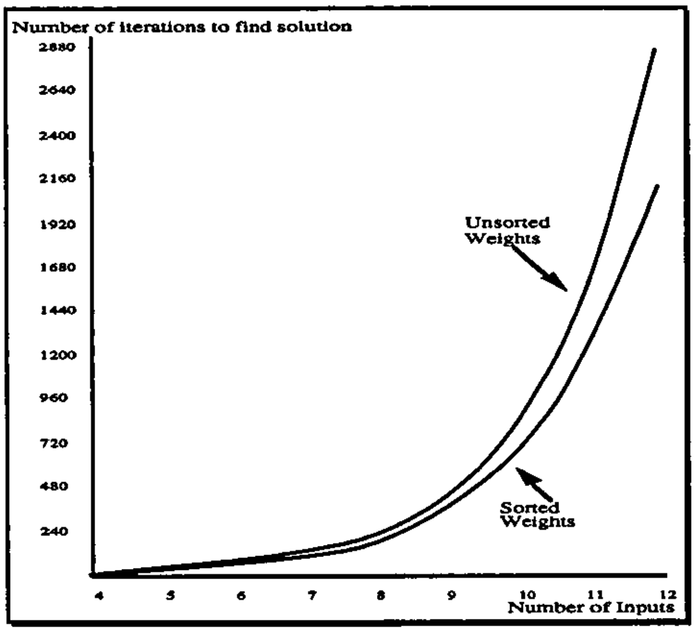


Figure 3.11. Comparing the speed of execution with sorted and unsorted weights. Although the lines appear close together the steepness of the curve masks a decrease in calculation time of 30 percent for the 12 input problem.

Examples of Boolean function derivation

Example 1

Consider the neuron in figure 3.12 representing a neural network implementation of a two input OR gate.

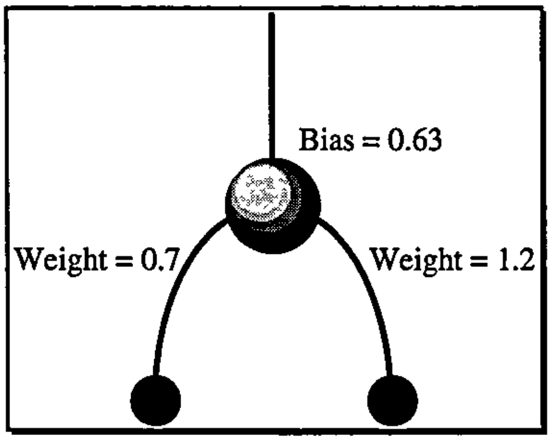


Figure 3.12 A two input OR gate implemented as a neural network.

This neuron implements an OR function. Applying the algorithm to find the minimum inputs to fire this neuron shows that either A or B on their own will cause the neuron to fire. Therefore the Boolean transfer function is

$$A \vee B$$

<u>Example 2</u>

A more complex example is the case of three dimensional parity. A neural network to calculate this function is shown in figure 3.13. The Boolean function for each neuron is calculated separately

$O = H$ $\qquad$ $H = G \vee F$

$G = \neg(\neg A \vee \neg B \vee \neg C)$ $\qquad$ $F = \neg(\neg D \vee \neg E)$

$D = \neg(A \wedge B) \vee \neg(A \wedge C) \vee \neg(B \wedge C)$ $\qquad$ $E = A \vee B \vee C$

combining these gives

$$\begin{aligned} O = \quad & \neg(\neg A \vee \neg B \vee \neg C) \vee \\ & \neg(\neg(\neg(A \wedge B) \vee \neg(A \wedge C) \vee \neg(B \wedge C)) \vee \neg(A \vee B \vee C)) \end{aligned}$$

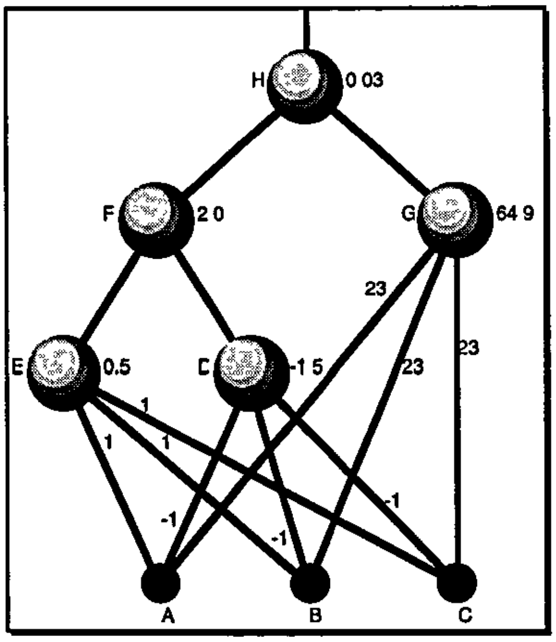


Figure 3.13. Implementation of three input parity as a neural network.

Application of Boolean rule extraction to rule induction on an adhesive dispensing machine controller

This application is the dispensing of adhesive in the manufacture of "mixed technology" P.C.B.s in which through hole and surface mount components are present on the same board. The surface mount components are secured to the board, prior to a wave soldering operation, by a small (0.0002 to 0.005 ccs depending on component) amount of adhesive. The amount of adhesive dispensed is critically dependent upon several process environment variables, (e.g. temperature, humidity, erratic thixotropic behaviour of the adhesive, air bubbles in the flow and variations in the P.C.B. substrate).

The dispensing unit consists of a syringe of adhesive coupled to a pressure control unit. The unit is made up of a solenoid valve, pressure regulator, temperature sensor and a pressure transducer to monitor the variation of pressure within the syringe. The dispensing unit is fixed to a SEIKO RT3000 robot which moves the syringe to locations of the P.C.B. where the adhesive has to be dispensed. Feedback data collection is carried out by an image processing system (Imaging Technology ITI151) coupled to a Pulnix TM-460 CCD camera incorporating a magnifying optical system.

The system consists of four processors (one SUN, two Target processors and one robot processor) with a pipeline vision processing system attached to the VME bus of the SUN. Communication between the software modules is either via the VME bus, serial line communications or the UNIX sockets mechanism. The "in-house" built Target was designed to allow the cell controller software, developed on the SUN to be ported to a "black box" solution suitable for an industrial system.

Messom et al. [1992] produced a neural network system for controlling the adhesive dispensing machine (figure 3.14). This is the sort of problem that has typically been tackled by the use of a rule induction package. The trained neural network should therefore be equivalent to a set of rules that could have been learnt by such a package.

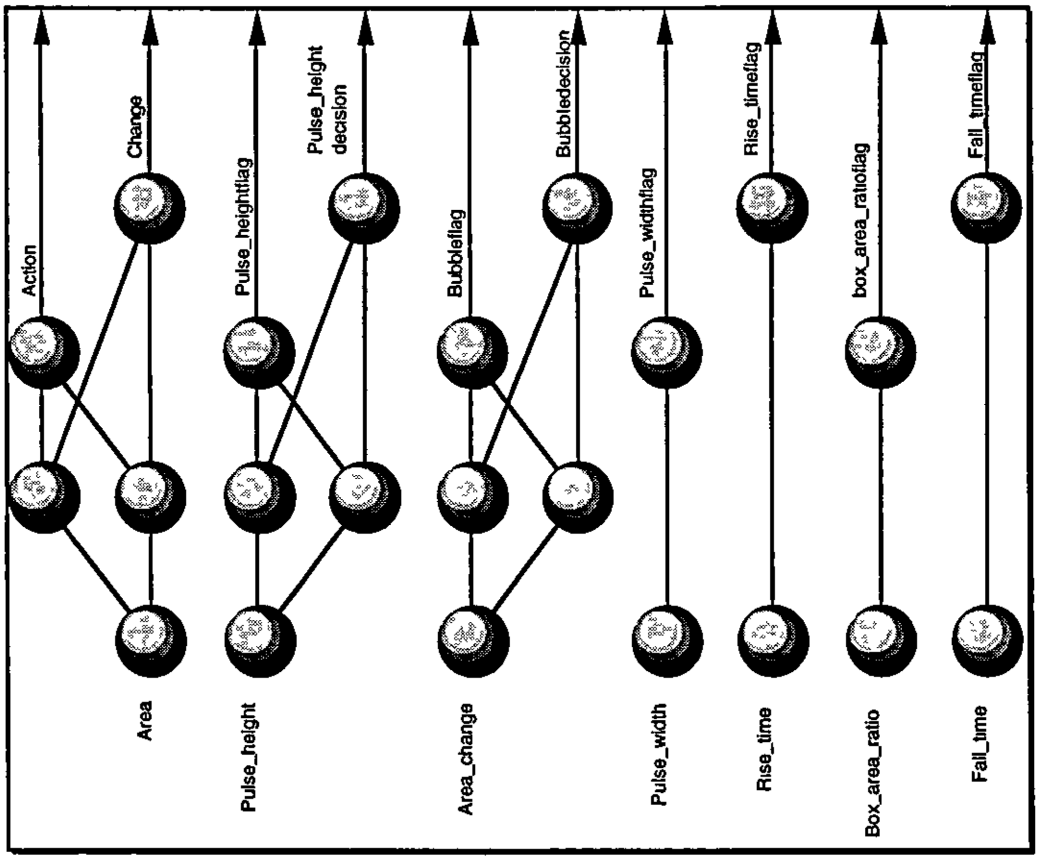


Figure 3.14. The net implemented by Messom et. al. for controlling a gluing machine.

The first layer of neurons receive real valued inputs, but produce outputs that are very nearly bi-polar. These neurons are effectively quantifying the input region, and are therefore called quantisation nodes. The actions of the quantisation nodes are described as a set of inequalities. This step does not simplify the information but does display it in a manner that is much more natural to read than a set of weights on a diagram. Examples of the rules resulting from these inequalities are show in figure 3 15.

IF rise_time <1.25	IF fall_time > 1.1
THEN rise-timeflag	THEN fall_timeflag
ELSE ¬rise_timeflag	ELSE ¬fall_timeflag
IF pulse_width < 1.0429	IF boxarearatio >
THEN pulse_widthflag	THEN boxarearatioflag
ELSE ¬pulse_widthflag	ELSE ¬boxarearatioflag
IF area > 1.0258	IF area > 0.9722
THEN midnode 1	THEN midnode2
ELSE ¬midnode 1	ELSE ¬midnode2
IF pulse_height < 1.025	IF pulse_height > 0.975
THEN midnode3	THEN midnode 4
ELSE ¬midnode3	ELSE ¬midnode4
IF area_change < 0.5500	IF area_change > 0.4499
THEN midnode5	THEN midnode6
ELSE ¬midnode5	ELSE ¬midnode6

Figure 3.15 The rules representing the first layer of neurons from the network in 3.14

The remainder of the network receives inputs that are close to bi-polar and delivers bi-polar outputs; as such they can be said to be implementing Boolean transfer functions. The results from expressing these transfer functions as rules are very similar in format to the first layer inequalities. Examples of the derived rules for the subsequent layers are shown in figure 3.16.

IF (midnode1 ∨ ¬(midnode2))	IF ¬((midnode1 ∨ midnode2))
THEN action	THEN change
ELSE ¬action	ELSE ¬action
IF ¬((¬(midnode4) ∨ ¬(midnode3)))	IF ¬((midnode3 ∨ ¬(midnode4)))
THEN pulse_heightflag	THEN pulse_heightdecision
ELSE ¬pulse_heightflag	ELSE ¬pulse_heightdecision
IF ¬((¬(midnode6) ∨ ¬(midnode5)))	IF ¬((¬(midnode6) ∨ midnode5))
THEN bubbleflag	THEN bubbledecision
ELSE ¬bubbleflag	ELSE ¬bubbledecision

Figure 3.16 The rules representing the second and subsequent layers of neurons from the network in 3.14.

It is now possible to directly implement this set of rules in a rule base without altering the action of the system. For clarity some general simplification of the Boolean rules is required, also substituting the inequalities produces much more natural looking results.

IF rise_time < 1.25
THEN rise_timeflag
ELSE ¬rise_timeflag

IF fall_time < 1.1
THEN fall_timeflag
ELSE ¬fall_timeflag

IF pulse_width < 1.0429
THEN pulse_widthflag
ELSE ¬pulse_widthflag

IF boxarearatio > 0.8520
THEN boxarearatioflag
ELSE ¬boxarearatioflag

IF area_change > 0.5500
THEN bubbledecision
ELSE ¬bubbledecision

IF (area > 1.02581 ∨ area < 0.9722)
THEN action
ELSE ¬action

IF (area_change > 0.4499 ∧ area_change < 0.5500)
THEN bubbleflag
ELSE ¬bubbleflag

IF pulse_height > 1.025
THEN pulse_heightdecision
ELSE ¬pulse_heightdecision

IF (pulse_height > 0.975 ∧ pulse_height < 1.025)
THEN pulse_heightflag
ELSE ¬pulse_heightflag

IF area < 0.9722
THEN change
ELSE ¬change

Figure 3.17 The complete set of rules derived from the network in 3.14.

The relative ease with which the transformation, from control network to rule set, can be made illustrates the usefulness of the interpretation system for medium sized control networks. This type of result is most useful to check that the hypothesis is reasonable, it also allows the network to explain its actions, something that classically they cannot do.

Noise Immunity of Neural Network Derived Rules

A good connectionist based solution should exhibit fewer overall noise connected problems than the currently available symbolic systems. A useful quantitative measure of noise immunity is the average percentage of the learnt hypothesis that matches the underlying training hypothesis. A correct function is represented by the truth table shown in table 3.1a, and a noisy version with one output incorrect is represented by table 3.1b. As these truth tables match in 87% of the outputs the noisy hypothesis is defined to have an integrity of 87%.

	I1	00000000	00000000
INPUTS	I2	01010101	01010101
	I3	00110011	00110011
	I4	00001111	00001111
OUTPUT		01010101	010101**1**1
		Table 3.1a	Table 3.1b

Table 3.1a is a truth table representing a Boolean function, whereas table 3.1b represents a noisy hypothesis. Note the change of the seventh bit in the output line (in bold).

Figure 3.18 shows the correctness of rules derived from networks that were trained to represent conjunction and parity, using examples containing different amounts of noise. Although live applications will normally be more complex than those illustrated, they will typically be trained on noisy data and will react in a similar manner.

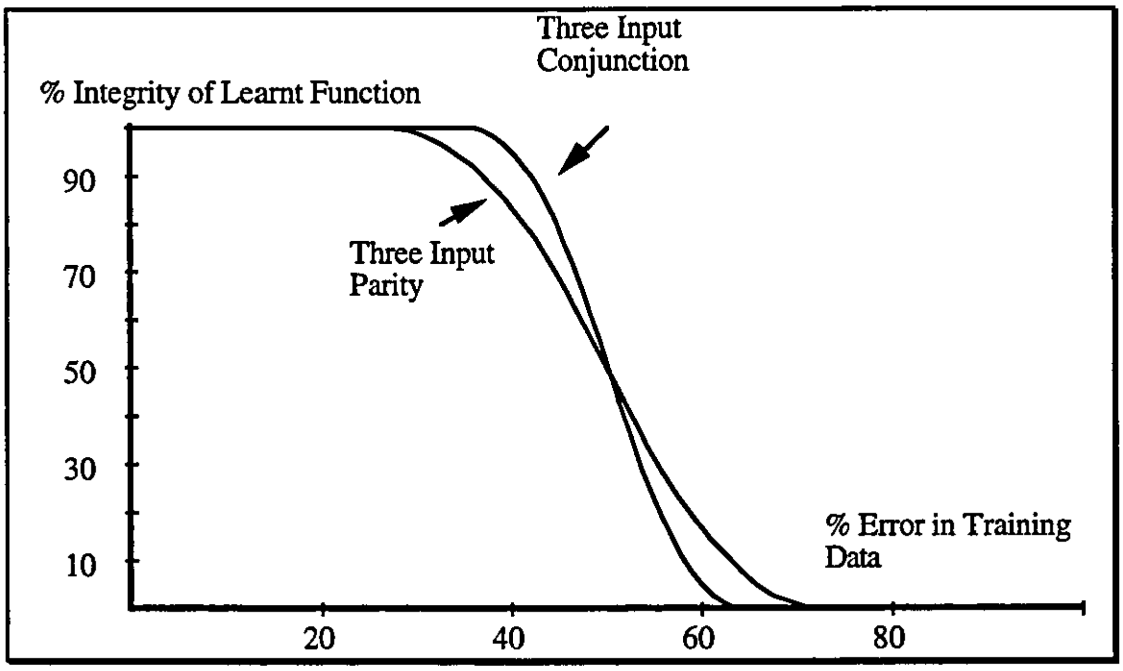


Figure 3.18 The response of a neural network derived from a 3 dimensional conjunction training set with differing amounts of noise, superimposed on the response of the neural network derived from 3 dimensional parity training set.

The Pole Balancing Problem

So far this chapter has argued for the extraction of Boolean rules from neural networks, and demonstrated how this may be accomplished. One of the major uses envisioned is the verification of the original neural network. The pole balancing problem is used to demonstrate the usefulness of the network derived rules in verifying the action of networks.

The pole balancing problem is an example of applied adaptive control and has become a standard tutorial problem. The control system must balance a pole on a motorised cart by moving the cart forward and back in a confined space. The implementation of most interest here is the neural network [Grant & Zhang 1989] shown in figure 3.19. This was successful in balancing the pole under a variety of circumstances. The inputs to this system are Boolean, so there is no quantisation layer and no need for inequalities in the rule set.

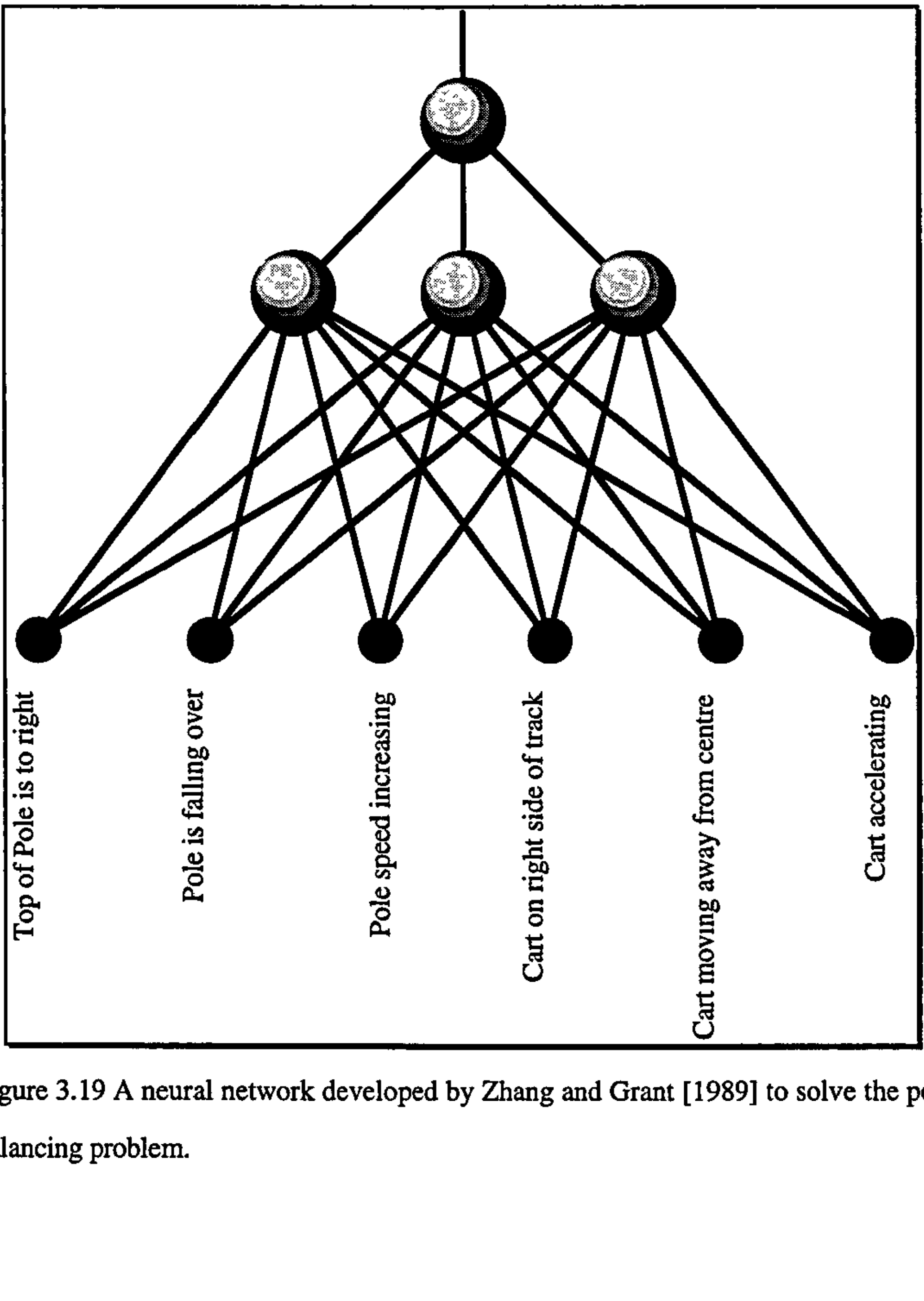


Figure 3.19 A neural network developed by Zhang and Grant [1989] to solve the pole balancing problem.

The analysis of the pole balancing net resulted in the following rule. Careful analysis of the rule will make it possible to find, and therefore correct, the errors in the network.

```
IF      (Top of pole is to right ∧ ¬Pole is falling over ∧
         ¬Pole speed increasing ∧ Cart accelerating)
                              ∨
        (Top of pole is to right ∧ ¬Pole is falling over ∧
         ¬Pole speed increasing ∧ Cart on right side of track ∧
         Cart moving away from centre)
                              ∨
        (Top of pole is to right ∧ Pole is falling over ∧
         Pole speed increasing)
THEN Apply right force
ELSE Apply left force
```

Figure 3.20. The rule resulting from an analysis of the network shown in figure 3.19

Simple image enhancement techniques can be used to simplify the rules, resulting in the underlying functions of the network and a list of exceptions. The underlying function can be verified and the exceptions indicate possible problems with the network hypothesis.

Applying simple image enhancement techniques to rules

The rule represented as a Karnaugh map in figure 13.21 has a fairly complex Boolean expression (figure 3.22), the map gives a better understanding of the simple basic concept. The aım of using image enhancement techniques is to allow high dimensionality problems to be represented in their original form but with much of the complexity removed. Enhancement will alter the rules, maintaining the main underlying objectives while removing superfluous small cases. This makes the analysis of the network hypotheses tractable for much larger networks.

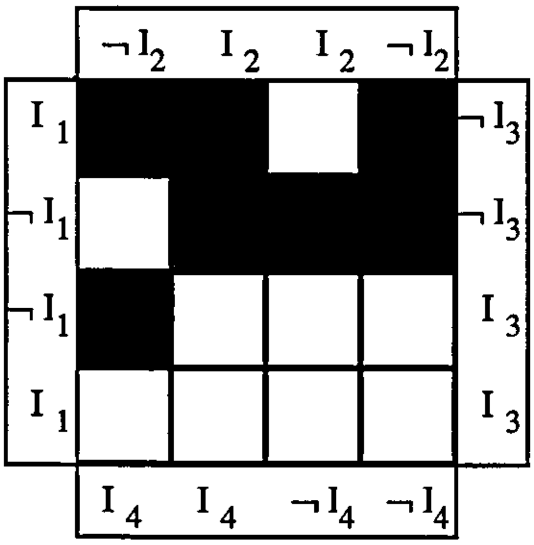
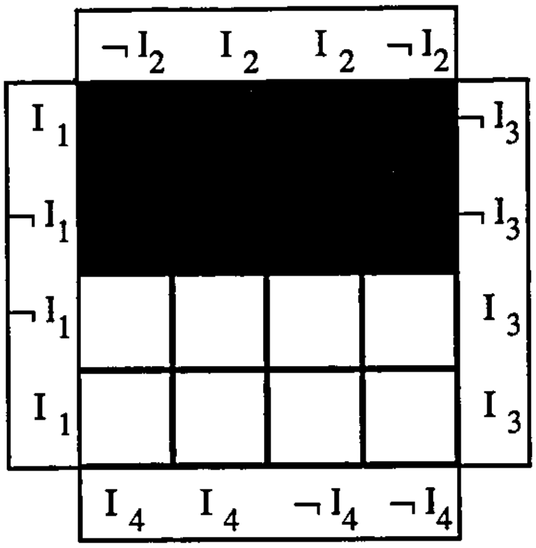


Figure 3.21 A function with a very simple basic concept but a relatively complex Boolean representation.

$$(I_1 \wedge I_3 \wedge I_4) \vee (\neg I_2 \wedge \neg I_3 \wedge I_4)$$
$$\vee$$
$$(I_2 \wedge \neg I_1 \wedge \neg I_3) \vee (\neg I_1 \wedge \neg I_2 \wedge I_3 \wedge I_4)$$

Figure 3.22 The expression represented in figure 3.21

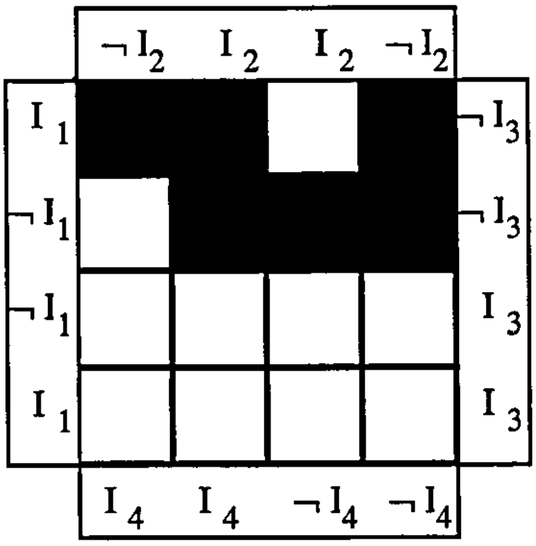
For example the Boolean expression represented in figure 3.21 could be simplified to become the one represented in figure 3.23.



$\neg I_2$ I_2 I_2 $\neg I_2$
I_1 $\neg I_1$ $\neg I_1$ I_1
$\neg I_3$ $\neg I_3$ I_3 I_3
I_4 I_4 $\neg I_4$ $\neg I_4$

Figure 3.23 An enhanced version of the Boolean rule in figure 3.21.

There are two separate processes required to enhance a Boolean rule. The first reduces it by removıng burrs. A burr is an area on the Karnaugh map that is separate from or smaller than the main body of the function. The example of 3.21 would become figure 3.24 after having its burr removed. The second process is the reverse, removing burrs from the inverse function fills any holes ın the rules. Hole filling on figure 3.24 would give the result shown in figure 3.23.



$\neg I_2$ I_2 I_2 $\neg I_2$
I_1 $\neg I_1$ $\neg I_1$ I_1
$\neg I_3$ $\neg I_3$ I_3 I_3
I_4 I_4 $\neg I_4$ $\neg I_4$

Figure 3.24 The example of figure 3.21 after having its burr removed.

The removal of burrs from a Boolean rule can be achieved easily if the Boolean expression is in a minimal disjunction of conjunctions. The example in figure 3.21 was represented as shown in figure 3.22. The Boolean representation for the version of the function shown in figure 3.24 after the burrs have been removed is shown below

$$\boxed{\begin{array}{c}(I_1 \wedge I_3 \wedge I_4) \vee (\neg I_2 \wedge \neg I_3 \wedge I_4) \\ \vee \\ (I_2 \wedge \neg I_1 \wedge \neg I_3)\end{array}}$$

Figure 3.25 The Boolean representation for the "after the burrs have been removed" version of the function shown in figure 3.24.

Each of the conjunctions in the Boolean equation represents an area on the Karnaugh map. The size of this area is inversely proportional to the length of the conjunction. By removing the longer conjunctions from the Boolean rule it is possible to remove the smaller areas on the map, thus removing the burrs from the rule. The Boolean equation in figure 3.22 is converted to the one in figure 3.25 by removing the longest of the conjunctions. Hole filling is an equally simple task. In effect hole filling is burr removal of the inverse function, which is exactly how it is implemented.

Using enhancement to explain the action of the pole balancing network

By using the simple image enhancement techniques on the rules derived for the pole balancing network it is possible to reveal information about the internal hypothesis. The rules for the network in their raw form were.

I1 - Top of pole to right	I4 - Cart on right side of track
I2 - Pole is falling over	I5 - Cart moving away from centre
I3 - Pole speed increasing	I6 - Cart accelerating

```
IF    $(I1 \wedge \neg I2 \wedge I3) \vee (I1 \wedge \neg I2 \wedge I4) \vee (I1 \wedge I3 \wedge I4) \vee$
      $(I1 \wedge I3 \wedge I5) \vee (I1 \wedge \neg I2 \wedge I6) \vee (I1 \wedge I3 \wedge I6)$
THEN
      M1 = True
ELSE
      M1 = False
```

```
IF    $(I1 \wedge \neg I2 \wedge I3)$
THEN
      M2 = True
ELSE
      M2 = False
```

```
IF    $(M1 \vee M2)$
THEN
      Apply right force to cart
ELSE
      Apply left force to cart
```

Figure 3.26 The rules for each of the neurons in the pole balancing network.

Repeatedly enhancing these rules gives the following three versions. Each is enhanced one step further than the preceding copy.

```
IF    $(I1 \wedge \neg I2 \wedge I3) \vee (I1 \wedge \neg I2 \wedge I4) \vee$
      $(I1 \wedge \neg I2 \wedge I6) \vee (I1 \wedge I3)$
THEN
      M1 = True
ELSE
      M1 = False
```

```
IF    $(I1 \wedge \neg I2 \wedge I3)$
THEN
      M2 = True
ELSE
      M2 = False
```

Figure 3.27 The rules for each of the neurons in the pole balancing network, where the longest conjunction has three elements.

```
IF      $(I1 \wedge \neg I2) \vee (I1 \wedge I3)$
THEN
        M1 = True
ELSE
        M1 = False
-----------------------------------------
IF      $(I1 \wedge \neg I2 \wedge I3)$
THEN
        M2 = True
ELSE
        M2 = False
```

Figure 3.28 The rules for each of the neurons in the pole balancing network, where the longest conjunction has two elements.

```
IF      I1
THEN
        M1 = True
ELSE
        M1 = False
-----------------------------------------
IF      True
THEN
        M2 = True
ELSE
        M2 = False
```

Figure 3.29 The rules for each of the neurons in the pole balancing network, where the longest conjunction has one element.

Substituting the simplest rules for M1 & M2 into the top level rule gives

```
IF Top of the pole is to right
THEN
        Apply right force
ELSE
        Apply left force
```

Figure 3.30 The simplest most basic rule for pole balancing.

The basic main rule is saying follow the top of the pole. As an expert in pole balancing you must agree that this is the correct basic rule. Secondly, and equally importantly, at a higher complexity we can check that the shape of the more important input variables are correct using a Karnaugh map. e.g. at an intermediate level of simplification the rule for applying force is

```
IF      (Top of pole is to right ∧ Pole speed increasing) ∨
        (Top of pole is to right ∧ Pole is falling over)
THEN
        Apply right force
ELSE
        Apply left force
```

Figure 3.31 A rule for pole balancing of intermediate complexity.

The map for this rule is shown in figure 3.32. The map shows that the rule is not symmetric. This means that the network will respond differently to the same situation on different sides. Although the network balanced the pole under test conditions it cannot be exactly correct. The reasons for moving the cart to the right should mirror the reasons for moving the cart to the left, so one of the rules must be wrong.

	$\neg I_1$	I_1	I_1	$\neg I_1$
I_3		■	■	
$\neg I_3$		■		
	I_2	I_2	$\neg I_2$	$\neg I_2$

Figure 3.32 The map of the rule after some simplification. This shows an asymmetrical response.

The part of the rule missed by the network corresponds to accelerating right when the top is to the left so as to slow down the movement of the pole if it is about to overshoot the centre line. It is not surprising this has been missed as it is not a common case. However, the training could be modified to force this example to occur.

Limits of Boolean rule conversion

The nature of neurons means that as the number of inputs grows so does the length of the Boolean description. In the worst case the number of conjunctions in the disjunctive normal form grows at a rate of 2^{n-1} where there are n inputs to the neuron. The time taken to derive a rule is proportional to the number of conjunctions. The rules become intractable to compute for neurons with more that 40 inputs, and meaningless for a human far earlier, see figure 3.33. A different method is needed to enable the network designer to interactively train and study the hypothesis of larger networks.

$$(ip20 \wedge ip19 \wedge \neg ip18 \wedge \neg ip17 \wedge \neg ip12 \wedge \neg ip11 \wedge ip10 \wedge ip9 \wedge ip4 \wedge \neg ip3 \wedge \neg ip2 \wedge \neg ip1)$$

or

$$(\neg ip26 \wedge ip25 \wedge ip19 \wedge \neg ip18 \wedge \neg ip17 \wedge \neg ip12 \wedge \neg ip11 \wedge ip10 \wedge ip9 \wedge ip4 \wedge \neg ip2 \wedge \neg ip1)$$

or

$$(\neg ip26 \wedge ip25 \wedge ip7 \wedge ip10 \wedge ip5 \wedge ip4 \wedge \neg ip3 \wedge \neg ip2 \wedge \neg ip22)$$

or

$$(\neg ip26 \wedge ip21 \wedge ip19 \wedge \neg ip18 \wedge \neg ip17 \wedge \neg ip12 \wedge \neg ip11 \wedge ip10 \wedge ip9)$$

or

$$(\neg ip21 \wedge ip13 \wedge ip12 \wedge ip11 \wedge \neg ip10 \wedge ip9 \wedge ip4 \wedge \neg ip3 \wedge ip2 \wedge ip1)$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

Figure 3.33 Showing the unintelligibility of the rules derived from a large character recognition network. This illustrates the difficulty of interpreting larger networks as Boolean functions. This Boolean rule has several thousand conjunctions, only the first few are shown here.

Analysing large optical character recognition networks

The method of describing the output function of a neural network should match the purpose for which the network was designed. The example being developed here is for the verification and testing of visual pattern recognition networks such as O.C.R. In these types of application the best form of verification is to get the network to draw what it thinks a typical input pattern should look like. If this matches with the developer's ideas then there is some justification for accepting the hypothesis as correct. Verification in this manner could be described as a form of feed backward network, trying to find the typical input firing pattern.

The ideal answer would be something along the lines of figure 3.34. The darker areas represent an area that has a stronger link with the output than the lighter areas. The hypothesis in figure 3.34 would be a good result from a network that has been trained to recognise the letter capital E. The figure 3.35 could represent a network that correctly classifies all the training examples but could do with some further training.

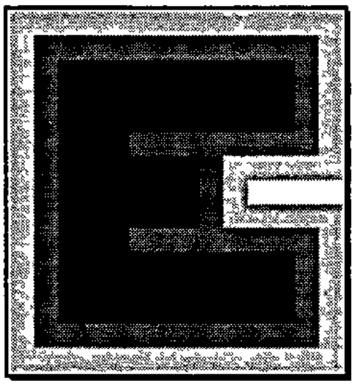


Figure 3.34 The target...... This is what a feed backward analysis of a network that has correctly learnt to recognise the letter E should look like. The darker the region the closer the link to the hypothesis.

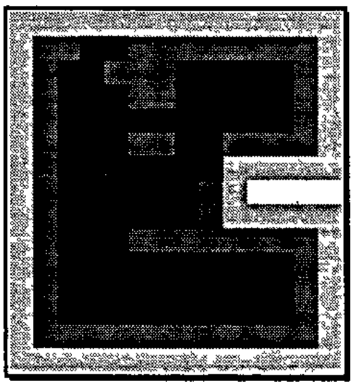


Figure 3.35 A feed backward analysis of a network that has correctly learnt to recognise the letter E. Some areas could do with further training examples.

The simplest way of finding the typical input patterns that cause firing is to experiment. While this is an excellent solution for small networks it soon becomes intractable for larger systems. Even with a simple O.C.R. network with 64 inputs there are $1.8*10^{19}$ different input patterns to test. This section of the chapter presents a tractable method for extracting a usable input pattern.

Finding the evidence for a neuron's output from its inputs

The output of a Boolean function is linked to each of its input parameters with different strengths. Given the function

$$\text{Output} = (I_1 \wedge I_2 \wedge I_3) \vee (I_2 \wedge \neg I_3 \wedge \neg I_4) \vee (\neg I_1 \wedge I_2 \wedge I_4) \vee (I_1 \wedge \neg I_2 \wedge \neg I_3 \wedge I_4)$$

represented by the map in figure 3.36 it is possible to say the output is strongly linked to I_2 with an index of 5, where the index is calculated as the number of positive outputs with I_2 TRUE, minus the number with $\neg I_2$ in the map. A complete list of the indices for this example is given in table 3.2.

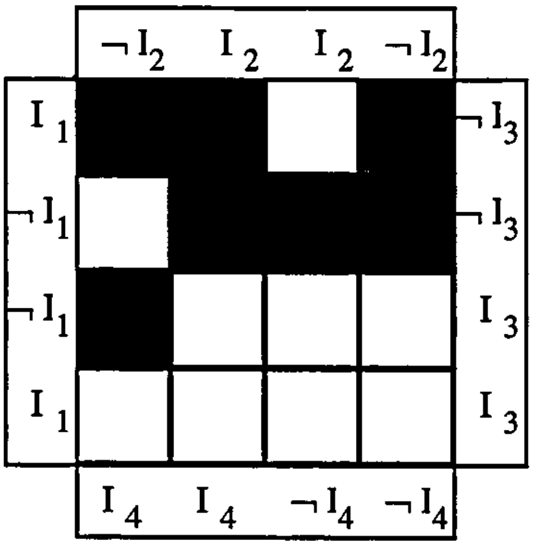


Figure 3.36 map of an example function

Input	Index
I_1	1
I_2	5
I_3	-1
I_4	1

Table 3.2 Indices of the different inputs to the function illustrated in figure 3.36

These indices uniquely identify any linearly separable function, however in this form they are not very useful for solving the defined problem. A better indicator is the proportion of evidence each input provides for identifying the function. This value is calculated as the proportion of the total indices attributed to each input. These figures have been calculated and shown in table 3.3

Input	Index	Evidence
I_1	1	0.125
I_2	5	0.625
I_3	-1	-0.125
I_4	1	0.125

Table 3.3. Proportion of evidence supplied by each input to the function in figure 3.29.

The evidence figure is the weight of importance attached to each input. This is a very similar idea to the original weight, raising the question why the original weights are not used. This is easily demonstrated with the two input conjunction implemented in the neuron shown in figure 3.37. As this is a conjunction each of the inputs is equally important in deciding the output but one of the weights is several thousand times the size of the other. Converting the neuron into a Boolean function and then calculating the evidences from that allows anomalies of this type to be removed.

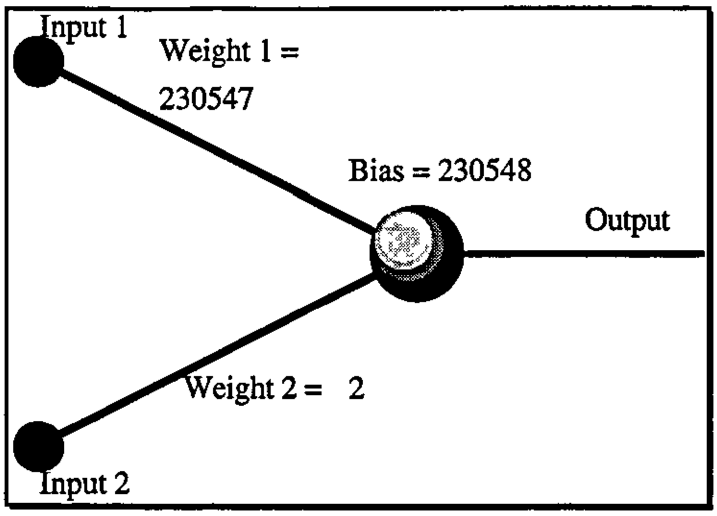


Figure 3.37 A simple conjunction implemented by a neuron with very uneven weights.

Feeding back the evidence for a networks output

Given the evidence that the inputs for each neuron provide for its output it is possible to combine the evidences to form global evidences for the final net output.

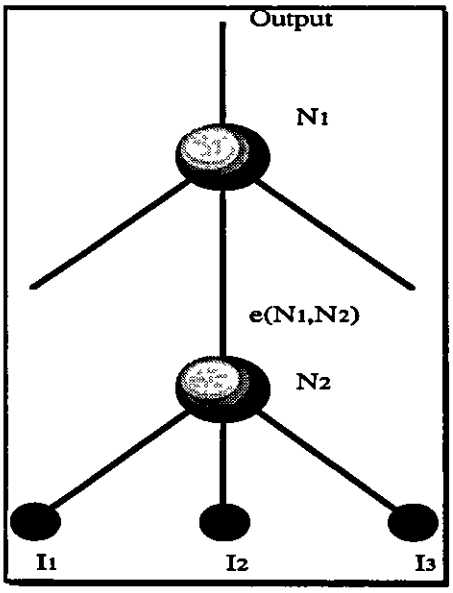


Figure 3.38 Part of a neural network illustrating the feeding backward of evidence.

Given two connected nodes N_o and N_i where the output of N_o is an input to N_i, some proportion of the evidence for the output of N_i can be attributed to the inputs of N_o. Let the evidence for the output of N_o attributed to the output of N_i be called

$$e(N_o, N_i)$$

Then the evidence E, provided by the Kth input to N_i for the output of N_o is given by

$$e(N_o, N_i) * e(\text{Input } K, N_o)$$

e.g. The evidence E for the output of N_1 given by I_2 in figure 3.38 is

$$E = e(N_1, N_2) * e(\text{Input } I_2, N_1).$$

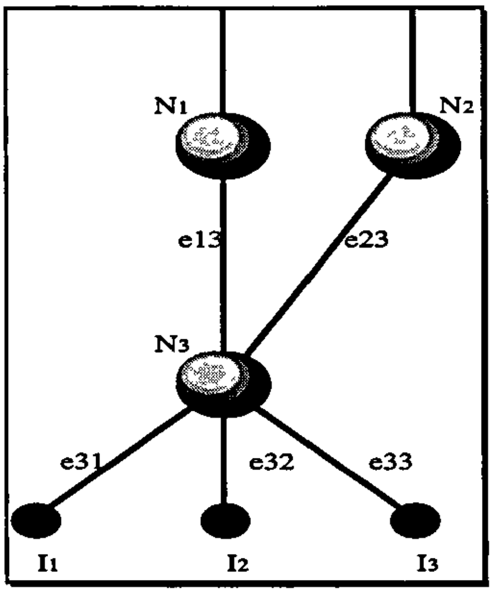


Figure 3.39 Part of a neural network illustrating the combination of evidences.

When the output of a node is connected via more than one route to the output then the evidence this output provides for the final output is the sum of the evidences on the inputs it is connected to. e.g. in figure 3.39 the evidence for the output given by N3 is calculated as e13 + e23. This makes the evidence provided by I1 = (e13+e23) * e31.

In general given a fully connected network (figure 3.40)

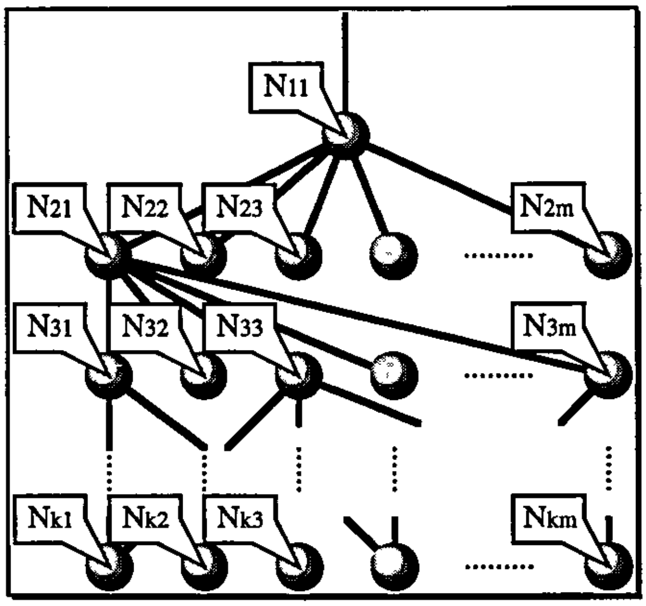


Figure 3.40 A fully connected network

Let the evidence for the final output passed between nodes N_{ij} and N_{lm} be represented by

$$e(N_{ij},N_{lm})$$

And the index of the input to node N_{ij} that comes from node N_{lm} be represented by

$$f(N_{ij},N_{lm})$$

Then the evidence supplied by the input of a node for the final output is

$$e(N_{ij},N_{(i+1)l}) = \frac{f(N_{ij},N_{(i-1)l})}{\sum_{1\ m}^{v} f(N_{ij},N_{(i-1)v})} * \sum_{1\ m}^{v} e(N_{(i-1)v},N_{ij})$$

Finding an approximation for the evidence quickly

The system so far illustrated for feeding back the evidences for a particular output value gives an accurate picture of the hypothesis within the connectionist network. The algorithm as it stands could be implemented, except that for nodes with a large number of inputs the function $f(X,Y)$ becomes intractable. This is the problem that the technique was developed to overcome. Therefore some approximation must be derived for the term :-

$$\frac{f(N_{ij},N_{(i-1)l})}{\sum_{1\ m}^{v} f(N_{ij},N_{(i-1)v})}$$

Looking at the equation represented in the Karnaugh map illustrated in figure 3.36

$$\text{Output} = (I1 \wedge I2 \wedge I3) \vee (I2 \wedge \neg I3 \wedge \neg I4) \vee$$
$$(\neg I1 \wedge I2 \wedge I4) \vee (I1 \wedge \neg I2 \wedge \neg I3 \wedge I4)$$

Any four of the conjunctions could cause this function to become true. Taking the first of these conjunctions.

$$(I1 \wedge I2 \wedge I3)$$

each of the terms can be said to supply $\frac{1}{2^3}$ of the total evidence to the output.

A good approximation of the total evidences is given by the summing evidence for each term over all the conjunctions in the minimal Boolean equation. If a term appears in a conjunction that has N terms, which in turn appears in a Boolean equation with M conjunctions then the evidence attributed to that occurrence of the term is

$$\frac{1}{M * 2^n}$$

This is calculated for each term in every conjunction in the Boolean equation and the total evidences are accumulated. Figure 3.34 shows a graph comparing the actual evidences between a node's inputs and outputs and those calculated by this approximate procedure for a set of randomly generated 5 input neurons.

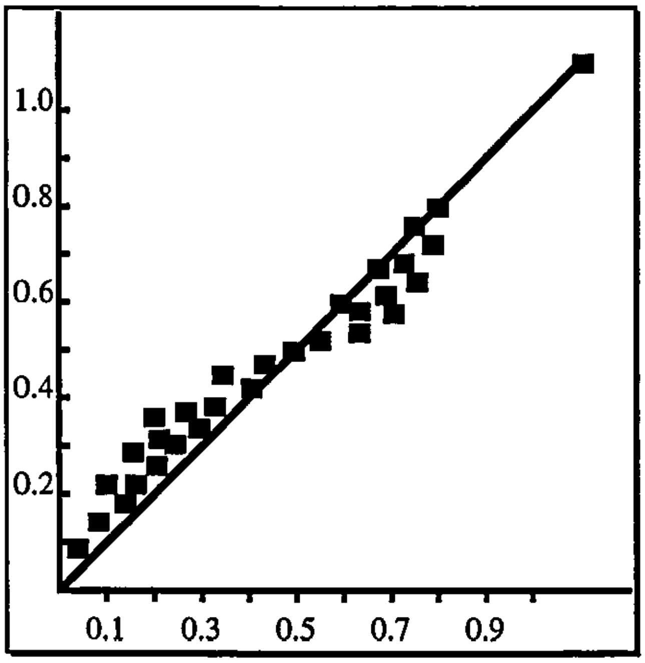


Figure 3.41 A graph showing approximate against actual evidences as calculated using the presented procedure for a set of five dimensional neurons.

While any approximation is not ideal the presented system finds a figure reasonably close to the actual value. This figure is good enough to produce acceptable results.

Example of tutorial O.C.R. neural network analysis

The first example, figure 3.42, has been carefully trained with a very expansive training set. This prevents any errors in the representation being attributed to the derived hypothesis. Figure 3.43 shows the training set used, this contains every combination of bars possible in a 5x4 grid an so should completely define the character. Only the first example is to be accepted as the letter E.

The net used contains 20 input nodes, 12 midnodes in a single layer and 1 output node. Each midnode is connected to 10 randomly selected input nodes and so the net, while being highly connected is not fully connected (figure 3.42).

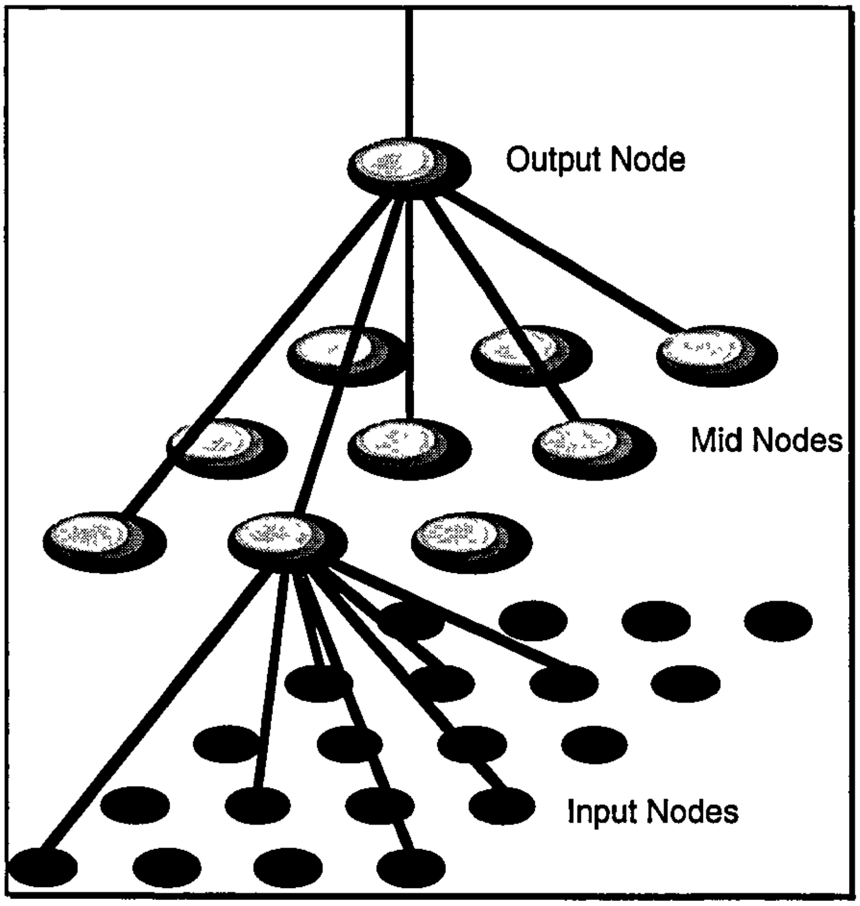


Figure 3.42 The small character recognition network used to produce the E hypothesis.

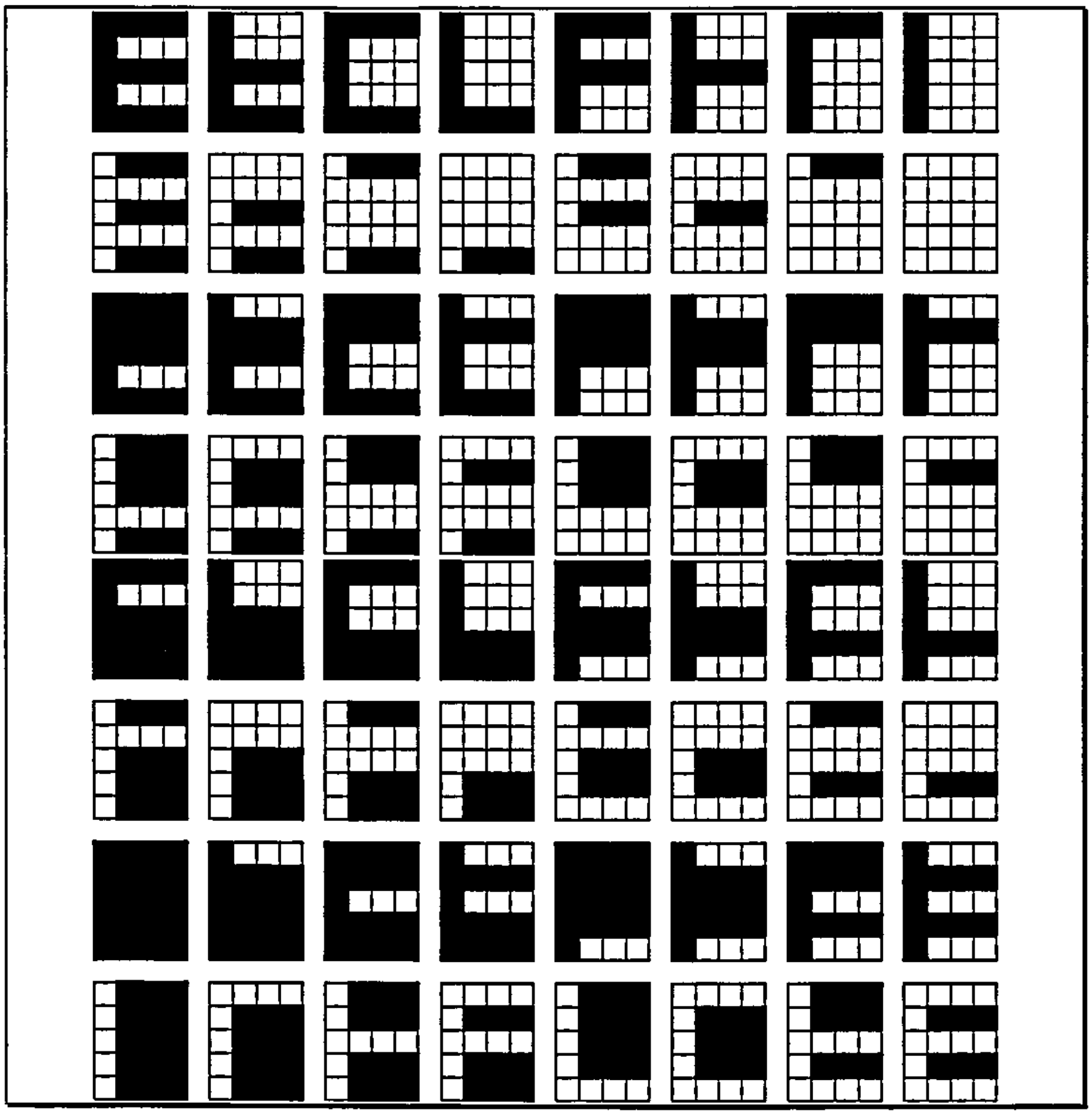


Figure 3.43 Training set used for the letter E.

Figure 3.44 shows the representation of the hypothesis derived by the approximation system. This shows that the latter has been faithfully represented by both the network and by the presented algorithm. This is strong evidence for accepting the system as correct.

The second demonstration example used the same network but with all the training examples reflected in the vertical axis, and then added to the originals. Thus giving two positive training examples of the letter E. One written forward and the other backward. The analysis of the derived network is shown in figure 3.45. The input nodes that are true in 50% of the positive training examples have been assigned approximately zero evidence for the output. This is correct as these nodes offer no evidence to determine if the input is the letter E.

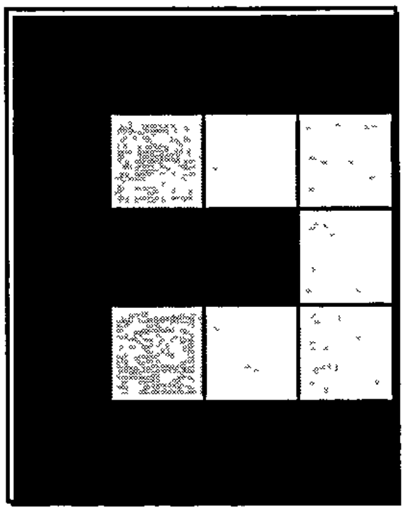


Figure 3.44 The analysis of a neural network that has been trained to recognise the letter E.

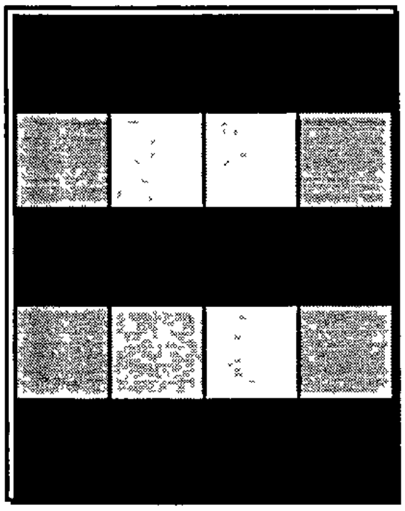


Figure 3.45 The analysis of a neural network that has been trained to recognise the letter E written either forwards or backwards.

Example of real O.C.R. network analysis

This example is also based on the letter E, but this time using a larger network trained on hand written images. This has been done with two separate independent subjects, the results of the analysis are shown in figures 3.46 and 3.47. The network used had 64 inputs, 12 midnodes each connected to 16 randomly selected input nodes and to the one output node.

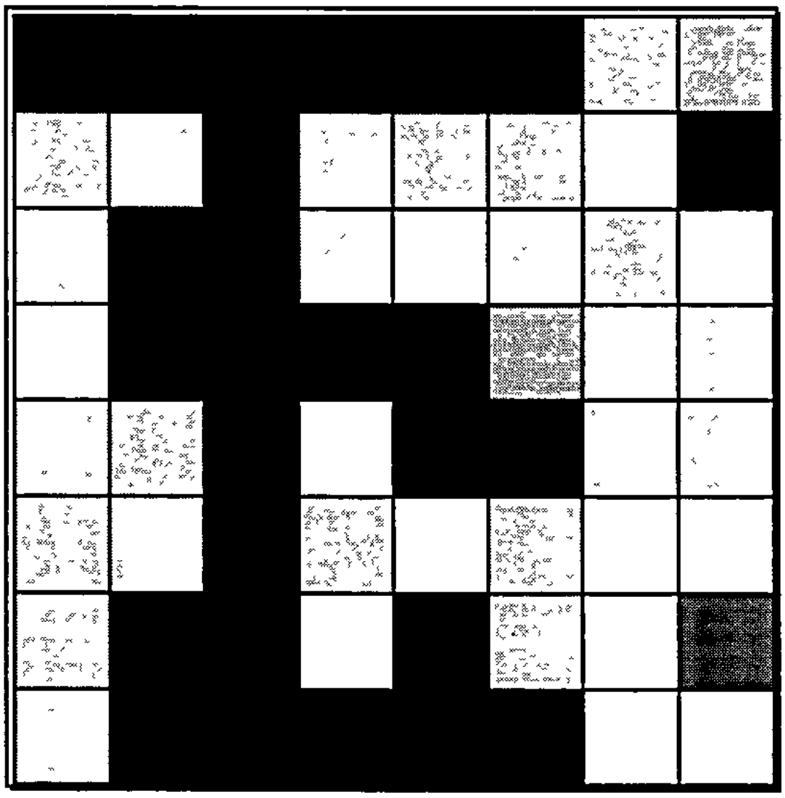


Figure 3.46 Hypothesis resulting from the first subject's hand-wrıtten E's.

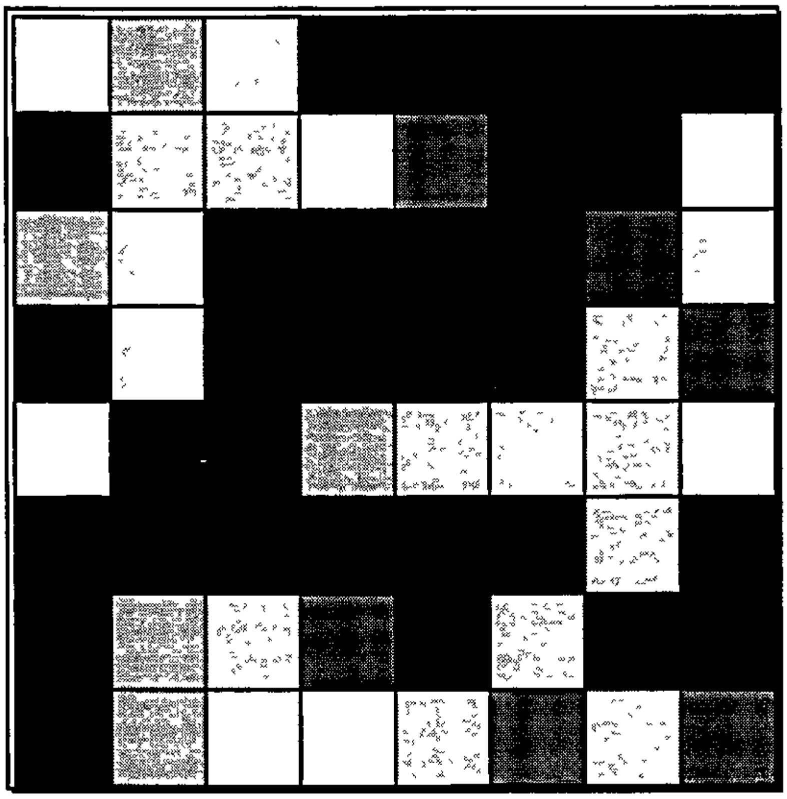


Figure 3.47 Hypothesis resulting from the second subject's hand-written E's.

Both subjects were asked for 12 versions of the letter E and 12 versions of objects not acceptable as the letter E. In both cases the networks were fully converged and classify their own training examples correctly.

It can be seen that in the first case the hypothesis formed has started to approach a correct answer, while in the second case a lot of further training will be required. This technique has demonstrated that the hypothesis formed by the first subject's hand writing is better than that formed by the second's. This is something that hitherto has not been possible.

Summary

The chapter has introduced some of the previous work aimed at representing neural networks as Boolean type functions. An analysis of one of these has shown a major representational flaw, indicating that simple Boolean functions are perhaps the best system to use.

A method for extracting the Boolean function from the weights and bias of a neuron has been presented. This algorithm has been expanded to make it faster and more flexible. The chapter also proved how to make sure that the answers delivered were minimal, allowing easier analyses and further increasing the speed.

The Boolean rules can be used to implement a rule base to replace the neural network controller, as illustrated by the adhesive dispensing machine, or to analyse and verify the hypotheses of the network (the pole balancing network). To aid the analysis of the Boolean function this chapter introduces the ideas of image enhancement to logical systems.

Finally the limitations of Boolean analyses are explored. Larger networks require a different system, the chapter closes by introducing a mechanism for interpreting large O.C.R. networks.

THE ACTIVATION FUNCTION
CHAPTER 4

Outline Of Chapter

The previous chapter demonstrated several methods and the major limitations for analysing neural networks. Perhaps the most important point to draw from the work is that if a network is properly designed then the analysis is easy. Badly designed networks produce complex and most often wrong hypotheses that become too big for accurate analysis.

The activation function inside the neuron is an important part of the network architecture. The sigmoid, in the guise of back propagation, was developed as a replacement for the threshold activation function to solve many of the then prevalent training problems [Bryson and Ho 1969, and Rumelhart et al 1986a,b]. Since this large and very significant step, very little work has been followed through to investigate the properties of other alternative activation functions. Most research aimed at improving the quality of the hypothesis or the speed at which it is obtained has concentrated on the topology of the network [Sietsma and Dow 1988, Marchand et al 1990, Frean 1990 and Mezard and Nadal 1990] or on enhancing the learning algorithm (normally back propagation) [Cater 1987].

The type of hypothesis that a network forms is heavily influenced by the activation function. Selecting the correct one has the same effect on analysis as selecting the correct topology; networks that use the correct activation functions are smaller and simpler than those that don't, and are therefore easier to analyse. This chapter looks at some of the properties of activation functions and at methods for dynamically selecting the correct one for individual neurons within feed forward networks during training.

Biological basis and other views

The sigmoid function is intended to be an approximation to a biological neuron's response. This is a large and wholly inaccurate assumption. [Griffith 1966] provides a simple explanation of the biochemical basis of neurons; "The cell is bounded by a surface membrane which is semi-permeable. This means that the membrane will allow certain ions and molecules to pass through it, but not others. The semi-permeability has certain consequences which can be understood using the theory of thermodynamics. One of these is that the interior of the cell normally has a negative electrostatic potential of about -70mV with respect to its exterior. However, if this potential difference is artificially altered above a threshold value of about -60mV, the membrane suddenly becomes permeable.... ...This change of permeability is self-propagating, it starts at one point and spreads to adjacent points and so on. Shortly afterwards the membrane recovers its semi-permeability. Thus a transient burst of electrochemical activity is generated."

The membrane of the cell covers the cell nucleus, but also extends down the axons to the dendrites. The axons and the synapses form the link between one nerve cell and its neighbours. Griffith further explains, "Once a cell has fired, the activity passes out to the uttermost extremes (the synapses), which then spew out small quantities of chemicals called transmitters. These pass across narrow gaps to adjacent cells where they alter the permeability of the membranes of those cells."

The potential difference across the cell membrane increases as quantities of the transmitter chemicals are absorbed. However the neuron does not fire until the potential difference reaches -60mV. The true activation function of a natural neuron is therefore a threshold function and not the sigmoid. For a close model of the activation of real neurons, asynchronous time dependant networks are required. The sigmoid should not be thought of as an approximation to biological systems, as the old threshold was a closer model. Given that the sigmoid is only an abstract activation function, developed to aid training, there is no reason why it should not be changed. The changes could be used to improve the accuracy and reliability of training and hypothesis.

Real neural networks model very complex information by using huge numbers of neurons, our own brain contains approximately 10,000 million neurons [Griffith 1966]. By developing a set of more complex activation functions it should be possible to produce better models of complex information, resulting in smaller and faster networks and the development of more powerful application areas for artificial neural networks.

Other recent work has presented the idea of parametrisation of the sigmoid function [Horejs & Kufudaki 1993]. By altering the parameters it is possible to affect the sigmoid's characteristics; i.e. such things as the maximum and mid values of the function. The idea was developed as a method of trying to remove the input weights as a training parameter, thus reducing the complexity in large networks. Each neuron is less powerful as it cannot affect its input weights, therefore this method results in massive networks of low complexity. The previous chapter argued that neural networks must be analysable, massive networks make analysing much harder as the information becomes more distributed. The aim should be to produce smaller not larger networks.

Sine as an Activation Function For Back Propagation

Back propagation relies on the use of a differentiable function as the activation function. The sigmoid was originally chosen because of its similarity to the then popular threshold activation function of the simple perceptron model [McCulloch and Pitts, 1943]. There is no reason why back propagation needs to use the sigmoid, replacing it with another differentiable function such as the sine function does not affect the theory of back propagation, but it does affect the types of hypotheses that can be represented.

Consider a single neuron with just two inputs (figure 4.1)

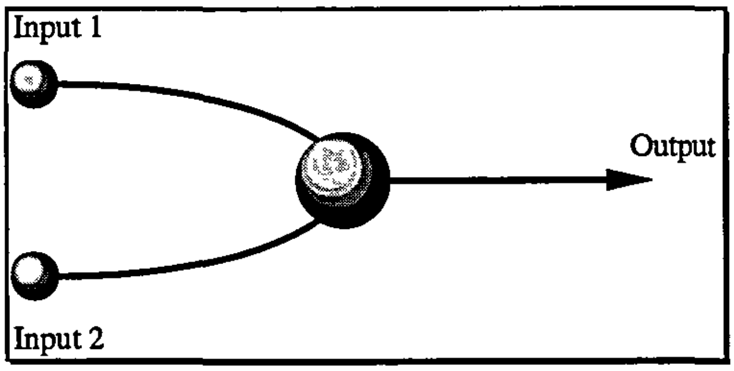


Figure 4.1 Simple two input neuron.

If this neuron uses a sigmoid as its activation function then it is perfectly able to model transfer functions such as "Boolean AND" and "Boolean OR", in fact it is difficult to imagine a better activation function for these two problems.

The "Boolean XOR" or two input parity is the simplest problem that cannot be modelled in a single neuron using a sigmoid activation function. The sine activation function is ideal for this problem. However, the sine activation is not very effective when trying to model "Boolean OR". Figures 4.2 a,b&c show how the sine function and the weights interact to give the "Boolean XOR" function. The dashed arrows represent the bias of the neuron and the solid arrows the input weights.

When both inputs are set to -1, the sum of the inputs outweighs the bias. The inputs sum to a negative value and the resultant output is negative.

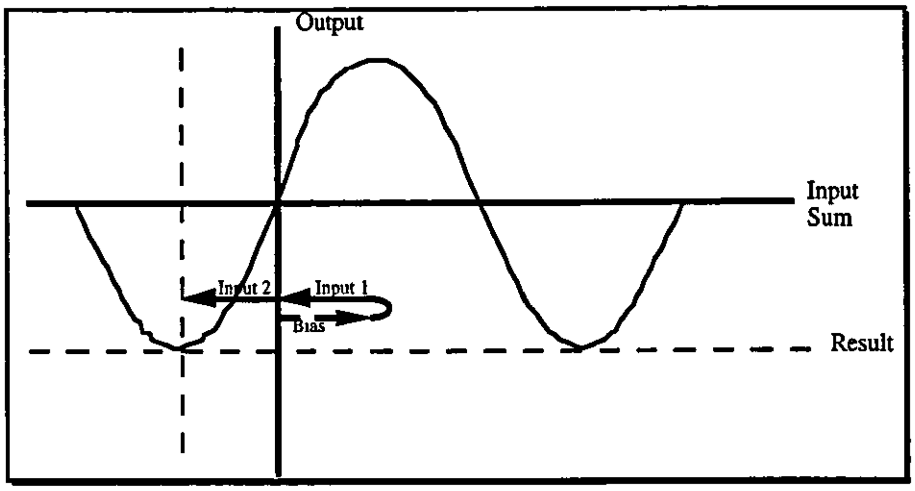


Figure 4.2a Demonstration of how the sine activation function and the correct input weights give the output for Boolean XOR when both inputs are FALSE.

If the inputs have different values and the input weights are the same then the inputs cancel out. The bias gives the correct input sum for a positive output.

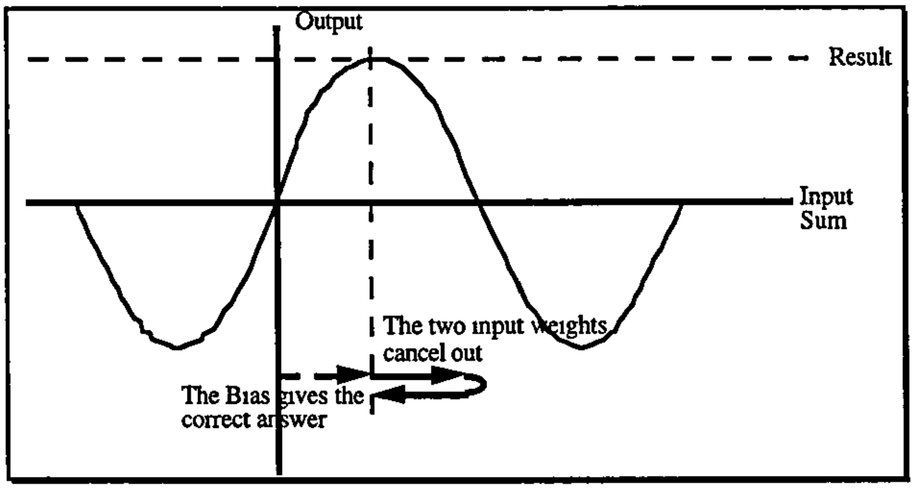

Output
Result
Input Sum
The two input weights cancel out
The Bias gives the correct answer

Figure 4.2b Demonstration of how the sine activation function and the correct input weights give the output for Boolean XOR when the inputs have different values.

When both inputs are true the input sum becomes larger, reaching the next trough in the sine function and therefore giving a negative answer again.

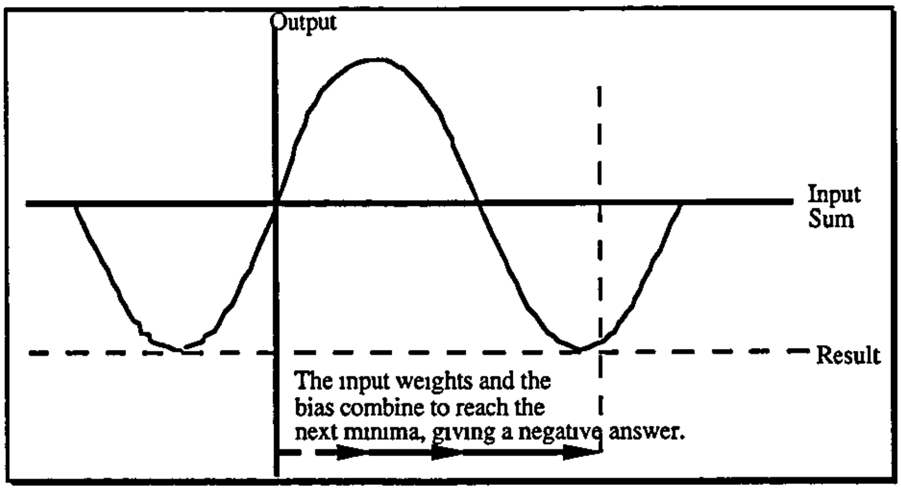

Output
Input Sum
Result
The input weights and the bias combine to reach the next minima, giving a negative answer.

Figure 4.2c Demonstration of how the sine activation function and the correct input weights give the output for Boolean XOR when both inputs are TRUE.

Each of the possible activation functions have different convergence characteristics. The advantages of using an activation function that easily models the required problem must be offset against the difficulties of achieving correct convergence with the back propagation training algorithm.

A Comparison of the Convergence Properties of the Sine and Sigmoid Activation Functions

The sine function produces a higher level of complexity in the back propagation algorithm. Consider the case of a single training example that is required to give an output value of zero.

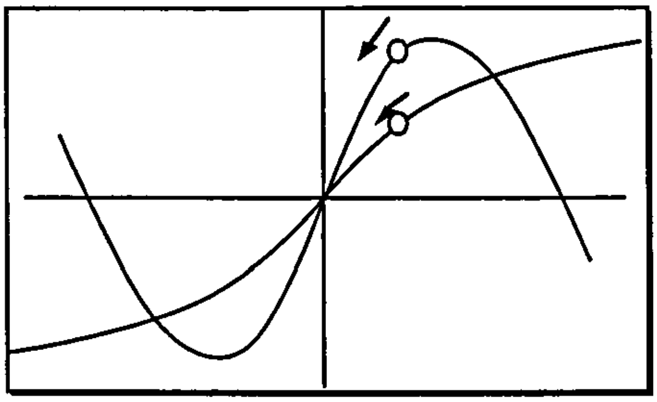


Figure 4.3 Comparison of convergence properties with one training example.

In both cases back propagation will reduce the size of the input vector to produce a closer answer. If the problem is now expanded to two training examples, they are both required to return zero as the correct answer.

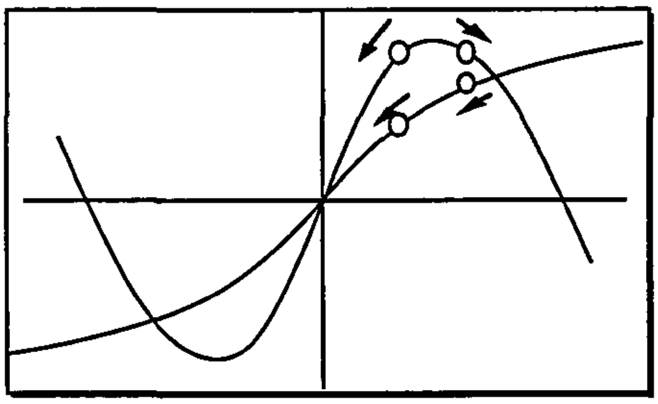


Figure 4.4 Comparison of convergence properties with two training examples.

Here the two examples are pushing in the same direction for sigmoid but are opposing each other when the sine activation function is used. This is because in the sigmoid's case there is only one possible direction for reducing the output values but in the sine function's case there are multiple points with the same output value. So when using a sine activation function the neuron must not only learn to give the correct answer for a particular training example but to use the correct cycle to allow correct learning of all the others.

In general a network using sine as its activation function contains many more local minima than a network that uses sigmoid. Therefore sine should not be used in a neuron when it is possible to model the data correctly using sigmoid, but there are problems that can be modelled only by adopting sine.

Selecting the correct activation function

Given a large network it is not feasible for the designer to allocate the correct activation function to every neuron. Therefore it becomes necessary to develop a "combined parametrised activation function" with enough flexibility to be able to model many different types of activation function. If the parameters of the general activation function are trained together with the input weights the neuron will develop the activation function that most easily models the data.

An example of a combined parametrised activation function can be constructed using both the traditional sigmoid and the sine activation functions. Let the activation function be $f(\beta)$ where β is the product of the weights and input values. i.e..

$$\beta = \text{bias} + \sum \text{weight}_n \,.\, \text{input}_n$$

$$f(\beta) = \propto.\text{sigmoid}(\beta) + (1-\propto).\text{sine}(\beta)$$

Each element of the activation function has a -1 to +1 range. The output of the neuron has been maintained in the traditional -1 to +1 range by making the sum of the activation function weights equal 1. By back propagating the errors and using them to modify the parameter 'a' in each neuron the individual neurons can be made to take on different characteristics. If $\propto$ is high, e.g. 0.9, then the activation function becomes close to the traditional sigmoid. This is very good when attempting to model problems like the "Boolean AND". Dropping the value of $\propto$ to 0.1 produces an activation function easily capable of representing the "Boolean XOR" problem.

Back propagation and the combined activation function

Back propagation assigns an error to each node in a feed forward neural network. In the output nodes this error is the deviation between the actual and required output values. In previous layers the error is a calculated share of the blame for the output error. Simply, the error assigned to each node can be thought of as showing whether it should have returned a higher or lower result. An explanation of exactly how back propagation assigns an error to each node can be found in chapter 1.

After the error on a neuron has been calculated, the weights assigned to its inputs are modified so as to make the error smaller and so bring the output slightly closer to the correct value. Let 'y' be the output of the neuron and '¥' be the error assigned to the neuron by back propagation. The amount a weight, W_n should change is given by

$$\Delta W_n = -\mu \cdot \frac{\partial y}{\partial W_n} \cdot ¥$$

Where μ is a small positive constant. By enumerating values for the variables it is possible to see how this update rule always creates a better answer.

If ¥ is positive then the output was too high. If $\partial y/\partial W_n$ is also positive then making W_n smaller will produce a lower and closer answer. The R.H.S. of the update rule becomes

$$\Delta W_n = \text{Negative} \cdot \text{Positive} \cdot \text{Positive}$$

ΔW_n is a negative number, therefore W_n is smaller after the update and the output is closer to the required value.

If ¥ is positive then the output was too high. If $\partial y/\partial W_n$ is negative then making W_n bigger will produce a lower and closer answer. The R.H.S. of the update rule becomes

$$\Delta W_n = \text{Negative} \cdot \text{Positive} \cdot \text{Negative}$$

ΔW_n is a Positive number, therefore W_n is bigger after the update and the output is closer to the required value.

If ¥ is negative then the output was too low. If $\partial y/\partial W_n$ is also negative then making W_n smaller will produce a higher and closer answer. The R.H S. of the update rule becomes

$$\Delta W_n = \text{Negative . Negative . Negative}$$

ΔW_n is a negative number, therefore W_n is smaller after the update and the output is closer to the required value.

If ¥ is negative then the output was too low. If $\partial y/\partial W_n$ is positive then making W_n bigger will produce a higher and closer answer. The R.H.S. of the update rule becomes

$$\Delta W_n = \text{Negative . Negative . Positive}$$

ΔW_n is a positive number, therefore W_n is smaller after the update and the output is closer to the required value.

The same update rule can be used with any adaptive parameter. The transfer function of the combined activation function was

$$y = \propto . \text{sigmoid}(\beta) + (1-\propto).\text{sine}(\beta)$$

where ß was the input value. The update rule for the parameter a is therefore

$$\Delta a = -\mu . \frac{\partial y}{\partial \propto} . ¥$$

Partially differentiating the transfer function with respect to a gives

$$\frac{\partial y}{\partial \propto} = \text{sigmoid}(\beta) - \text{sine}(\beta)$$

Therefore with each training iteration the parameter '∝' can be updated along with all the other adaptive input weights using an identical method.

Examples of training with the combined activation function

All two dimensional Boolean problems can be represented using just a single neuron with a combined activation function. To demonstrate the neuron's ability to derive the correct activation function, the solutions for "Two input AND" and "Two input XOR" are presented.

Two Input AND

	Example	Number		
	1	2	3	4
Input 1	-1.0	+1.0	-1.0	+1.0
Input 2	-1.0	-1.0	+1.0	+1.0
Output	-1.0	-1.0	-1.0	+1.0

Figure 4.5 The set of training examples used for two input AND

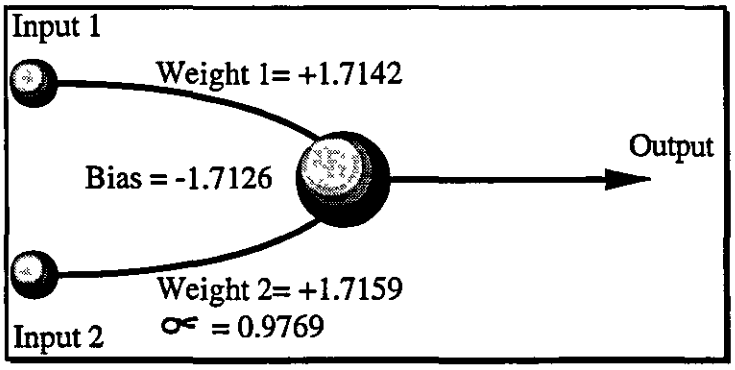


Figure 4.6 The neuron derived using back propagation for two input AND

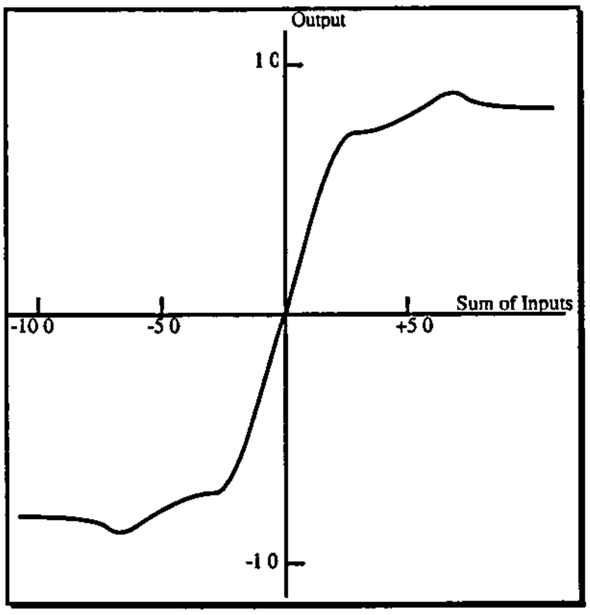


Figure 4.7 The activation function of the neuron derived using back propagation for two input AND

Two Input XOR

	Example	Number		
	1	2	3	4
Input 1	-1.0	+1.0	-1.0	+1.0
Input 2	-1.0	-1.0	+1.0	+1.0
Output				

Figure 4.8 The set of training examples for two input XOR

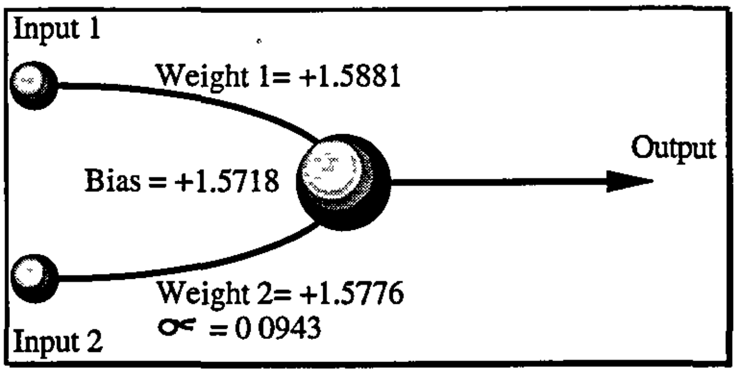


Figure 4.9 The neuron derived using back propagation for two input XOR.

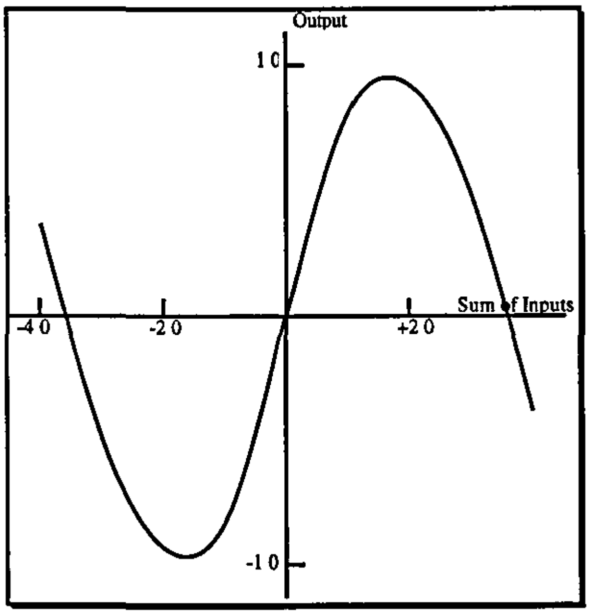


Figure 4.10 The activation function of the neuron derived using back propagation for two input XOR.

Both the neurons have developed activation functions capable of representing their own training sets; one has produced a threshold type function and the other a cyclic function, ideal for the two different problems.

The two very simple examples show that neurons which use a combined activation function are more flexible in the representations they can adopt. If individual neurons are more flexible then networks built using these neurons must also be more flexible. If it is easier to represent information within the network, the time required to find an acceptable representation should be reduced. Below is a four input, three midnodes and one output network that is the basis for an experiment designed to demonstrate the improvement in training times achievable using a combined activation function.

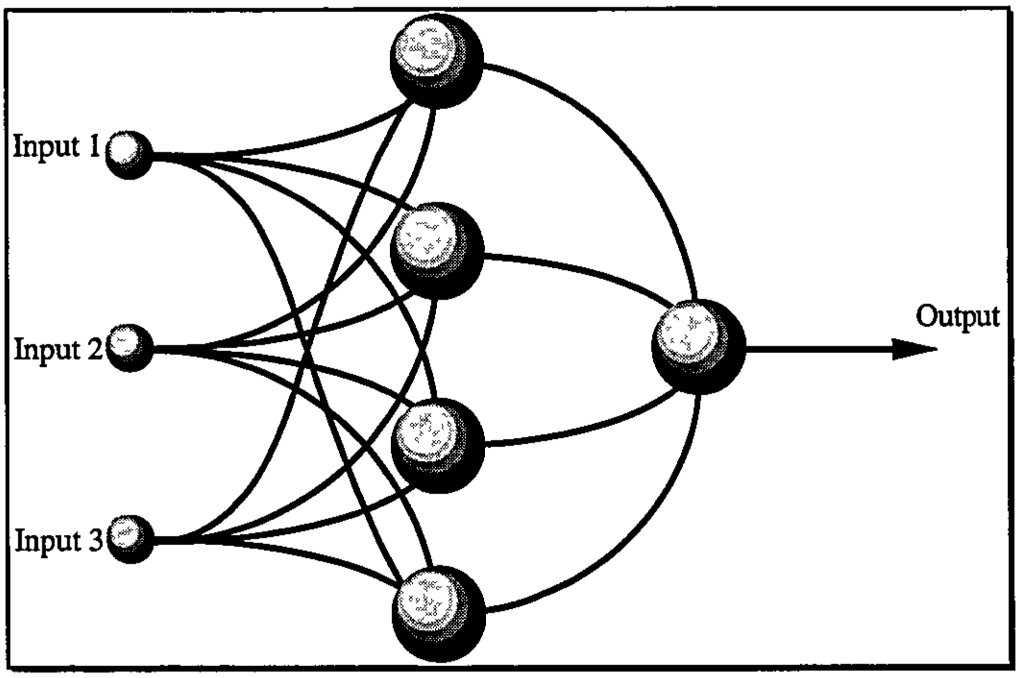


Figure 4.11 The network used for 3 input training experiments.

In this example 20 randomly chosen 3 dimensional functions were learnt by the network shown above. The graphs show the average of the sum of squares error over these 20 training sets for both the traditional and combined activation function. The same training sets and starting points were used for both experiments.

A sum of squares error of zero shows a perfect hypothesis; this is practically unobtainable. If the sum of squares error is below one then every example is on the correct side of zero, allowing thresholding to produce the correct answers. It is normal to attempt a sum of squares error of approximately 0.1 as this moves all of the examples away from zero allowing a middle region of 'Unknown'.

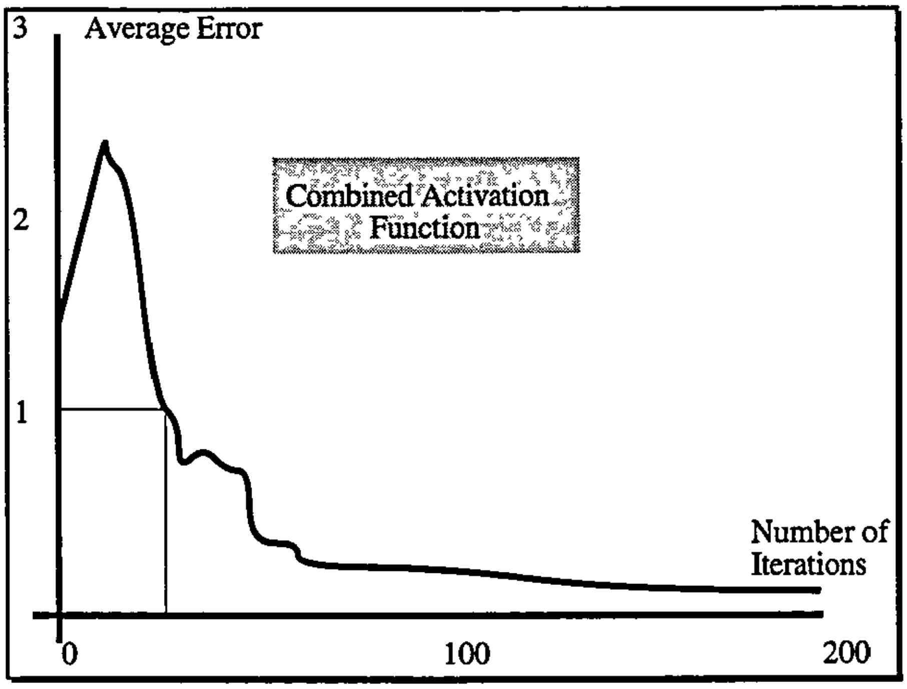


Figure 4.12 Average sum squared error when using a combined activation function

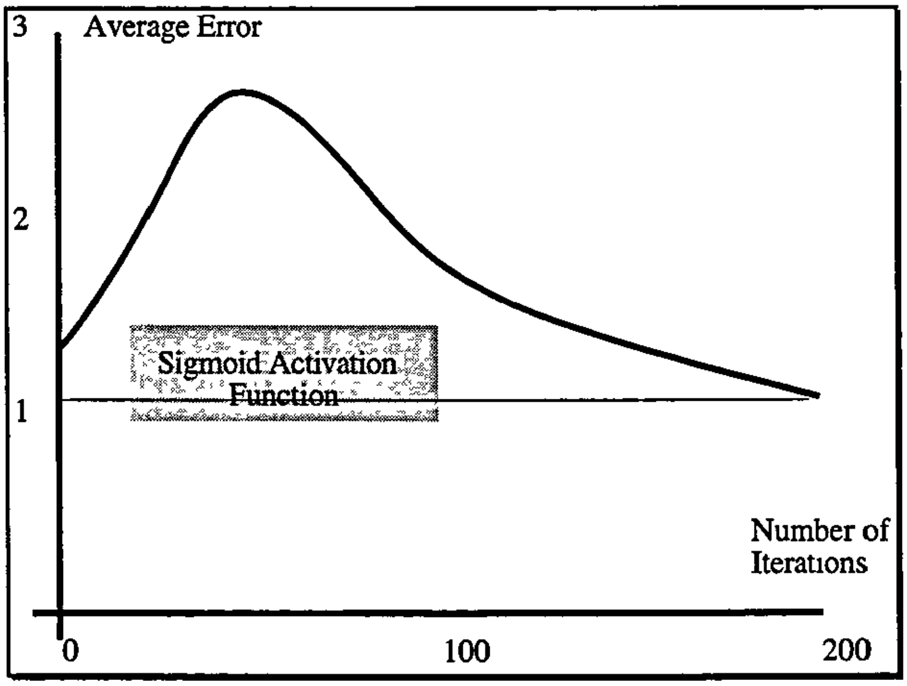


Figure 4.13 Average sum squared error when using a traditional sigmoid activation function.

The results show the potential for very large savings in training time. The combined activation function reaches an error of below 1.0 (all examples correct) approximately six times faster than the traditional sigmoid activation function.

The Colour Modelling Problem

To further demonstrate the added powers of the combined activation function, a larger industrial application is presented. Here the problem is to model the way the human eye perceives colour in different ambient lighting conditions, and when printed on different materials.

It is well known that perception of colour is different under different conditions, viewing angle, lighting and texture are just some of the aspects that contribute towards different perceptions of the "same" colour. This presents problems for many people, catalogues for example which show a different colour in their pages from that of the garment or product itself.

The complexity of predictive colour modelling has exercised researchers for some time. For the purposes of the research reported here we have been given a data set to determine Colourfulness, Brightness and Hue from the basic tristimulus values under constant viewing conditions. The data were derived from a well researched model of colour perception and based on the work of Luo et al. [1991] and Wang and Malakooti [1992]. The science of colour perception dates back to the 19th century with work by Young [1802] and later by Helmholtz [1924]. This work gave rise to the theory that only three colour receptors existed in the human eye, red, green and blue. More recent models by Hunt [1987] are based on a simplified theory of colour vision for chromatic adaptation together with a uniform colour space.

We are grateful to Luo and Wang for supplying the data without which the subsequent development of the neural network based models would not have been possible.

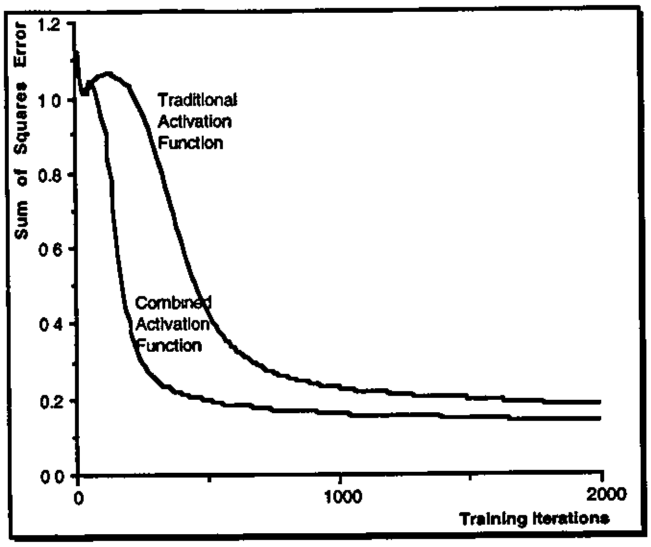


Figure 4.15 The errors while training a network with two hidden nodes

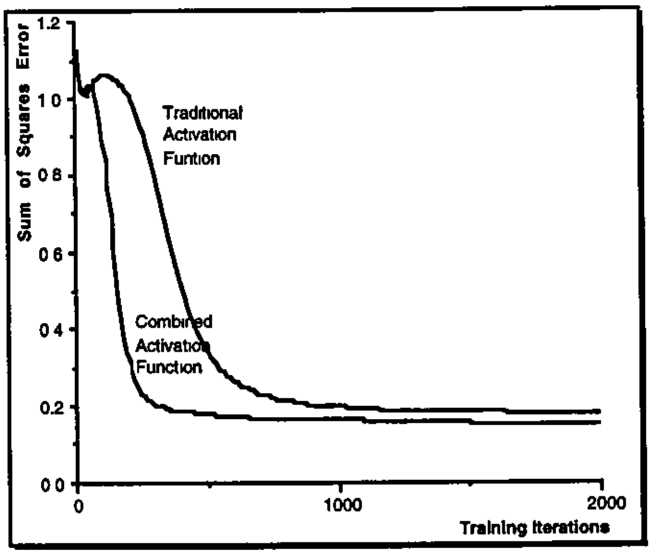


Figure 4.16 The errors while training a network with four hidden nodes

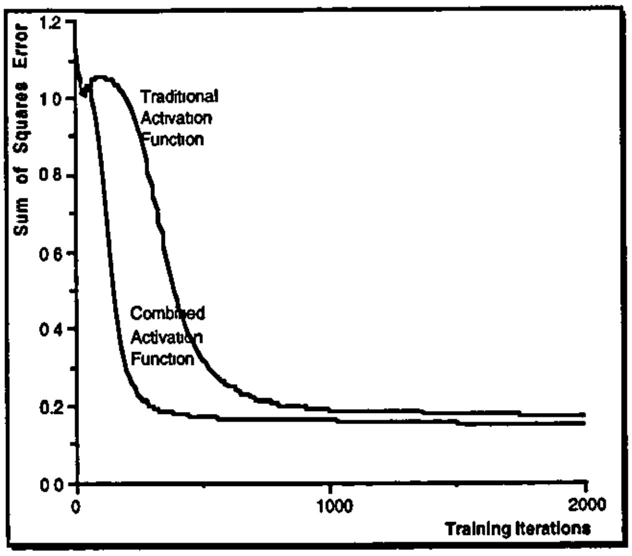


Figure 4.17 The errors while training a network with six hidden nodes

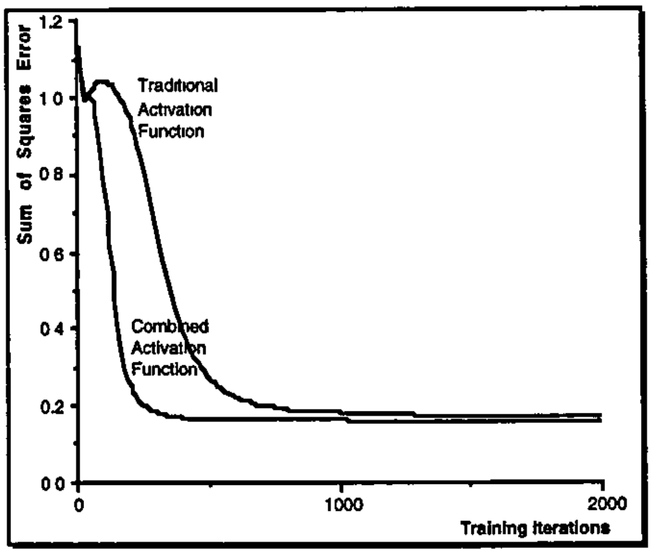


Figure 4.18 The errors while training a network with eight hidden nodes

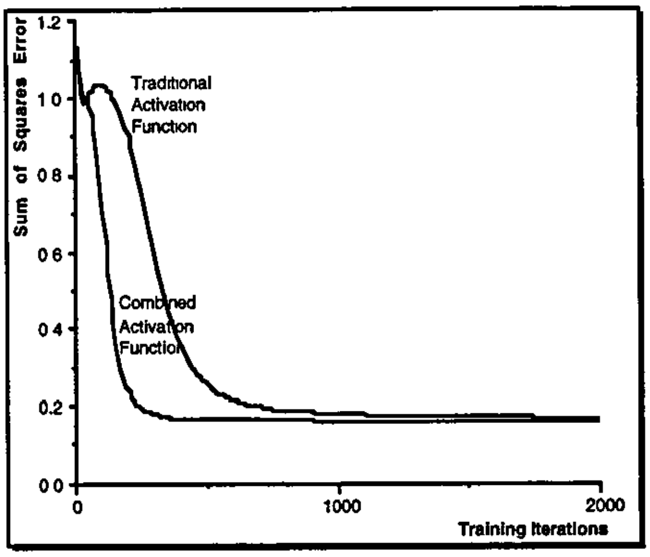


Figure 4.19 The errors while training a network with ten hidden nodes

In each graph the quality of the answers obtained by the combined activation function are very similar after 2000 training iterations. However, the combined activation function is achieving these results much faster than the traditional one. Figures 4.20 and 4.21 demonstrate the observed acceleration in learning. The presented graphs show that a significantly smaller number of iterations are required to reach the target error tolerances, resulting in much faster training.

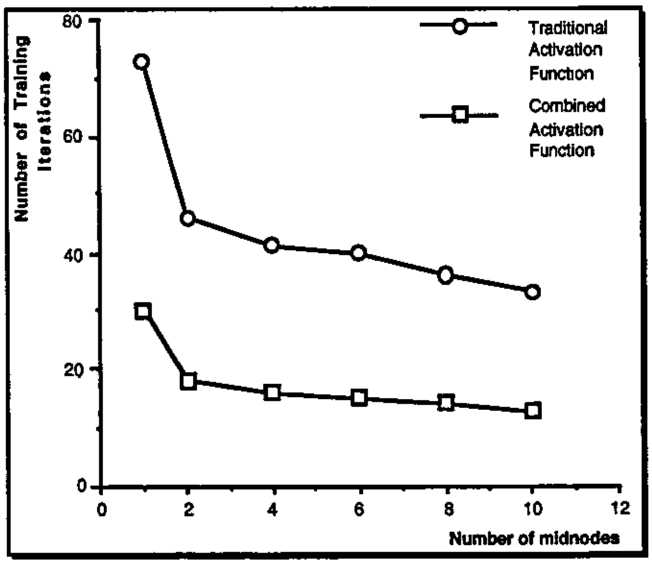


Figure 4.20 The number of training iterations required to achieve a sum of squares error of 0.5 for different sizes of hidden mid layer.

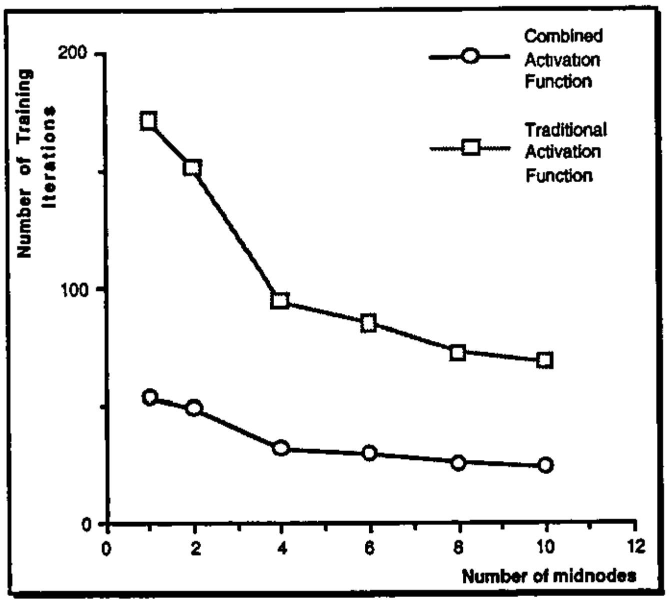


Figure 4.21 The number of training iterations required to achieve a sum of squares error of 0.2 for different sizes of hidden mid layer.

As well as increasing the speed of convergence, swapping to the combined activation function vastly reduces the size requirement of the network. The peak performance of the networks varies with the network size; getting the correct size of network results in a better answer than one provided by a smaller or larger network. By looking at the final error it is possible to find the best network size for the different activation functions (figure 4.22). This shows that the best size for the combined activation function is 2 midnodes and that for the traditional activation function it is more than ten.

Moving to the combined activation function has allowed a neural network with only two neurons in its middle layer to model data that would have used more than ten traditional neurons. The training is faster and the final result closer to zero error.

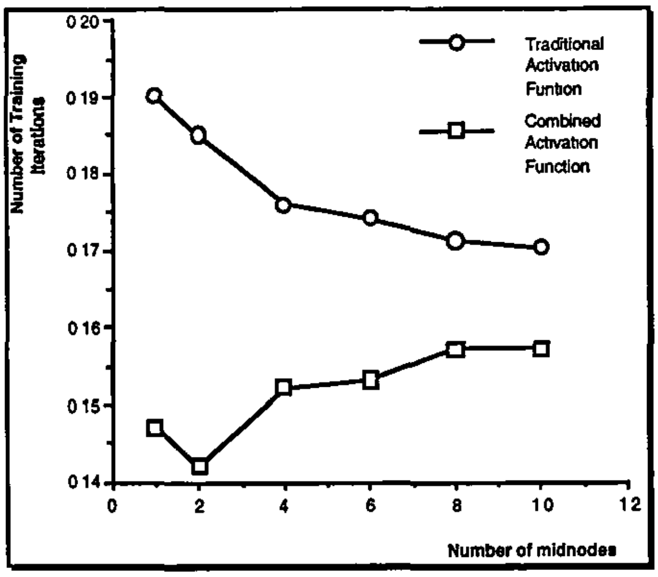


Figure 4.22 The final error for different network sizes after 2000 training iterations.

Summary

This chapter has looked at the activation function within the neurons that build up neural networks. By selecting the correct activation function it is possible to vastly reduce the size and complexity of the network required for a particular problem; demonstrating the similarities in effect between selecting the correct activation function and correctly designing other aspects of the architecture.

As with all changes there are problems as well as advantages. While it is easier to model certain hypotheses using non-standard activation functions they can be very difficult to train, making the correct choice of activation function a problem of great importance.

The combined activation function was presented as a method of selecting an activation function. Selection works by allowing each neuron to dynamically change its own activation function during the training process. This is achieved by extending back propagation to cover the activation function as well as the weights that interconnect neurons.

A more general neuron gives a more powerful neuron. By increasing the power it is possible not only to reduce the size of the networks, but also to increase the training speed. Neurons using a combined activation function train faster and use smaller networks. The previous chapter showed that network size was the limiting factor when analysing the hypotheses. The production of an analysis technique for the combined activation function could therefore allow analysis and verification of more complex systems. The production of an analysis tool for the more complex neuron presents one path of possible continued research.

Training for continuous models consists of calculating the correct output values for a set of example inputs and then using back propagation in the same manner as for classification problems. A simple tutorial problem is that of calculating the distance a ball will travel if thrown at a fixed velocity but at a variable angle[1]. The figure below shows how the angle at which the ball is thrown affects the distance it will travel.

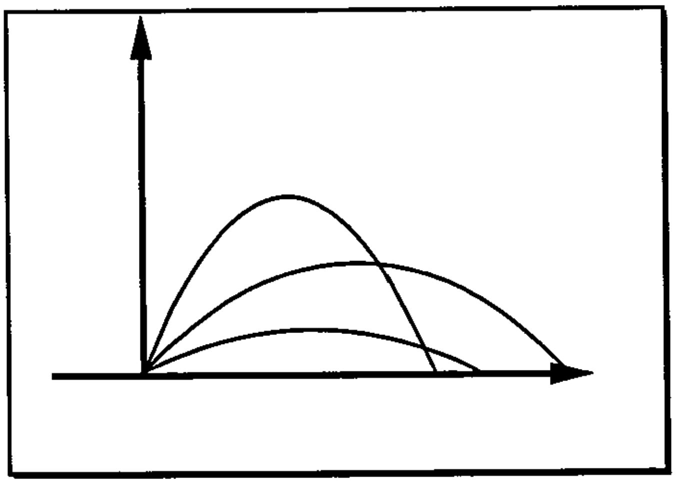


Figure 5.1 : The trajectory of a ball thrown at different angles but with a common speed. Note the different distances that the ball travels.

Angle from Horizontal	Distance to first bounce
5	0.1739
15	0.5000
25	0.7671
35	0.9404
45	1.0000
55	0.9404
65	0.7671
75	0.5000

Figure 5.2 The different distances that the ball travels

1 The figures used as training examples assume the experiment takes place in a vacuum, $g = 10.0\ ms^{-1}s^{-1}$ and that the initial speed of the ball is $\sqrt{10}$.

Problems with Continuous Modelling

Given that we require the correct answer to within a fairly small tolerance then a neuron has a small range of acceptable output values. The tolerance on the input vector must become very tight to meet this requirement. Figure 5.4 shows that requiring a real answer within even a fairly loose tolerance results in a very tight input tolerance. A tight tolerance on the input vector makes the values assigned to the input weights critical. If exactly correct values are required for the weights, training becomes a much harder, and therefore longer, problem.

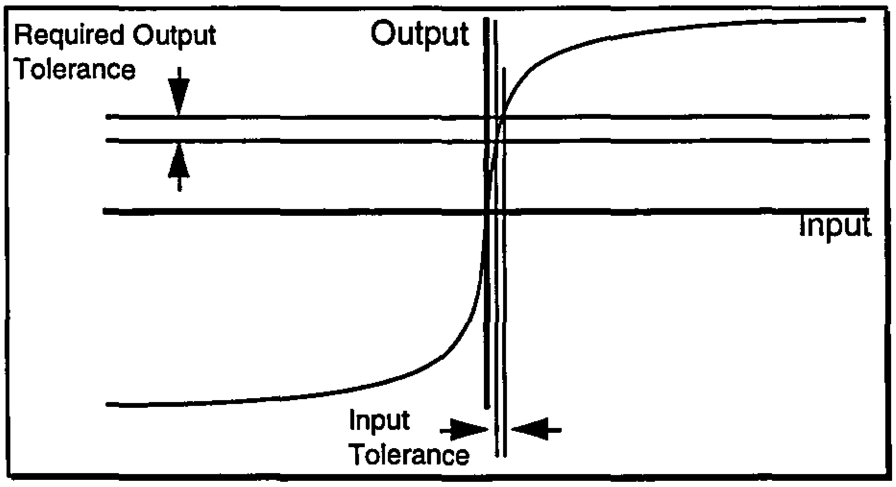
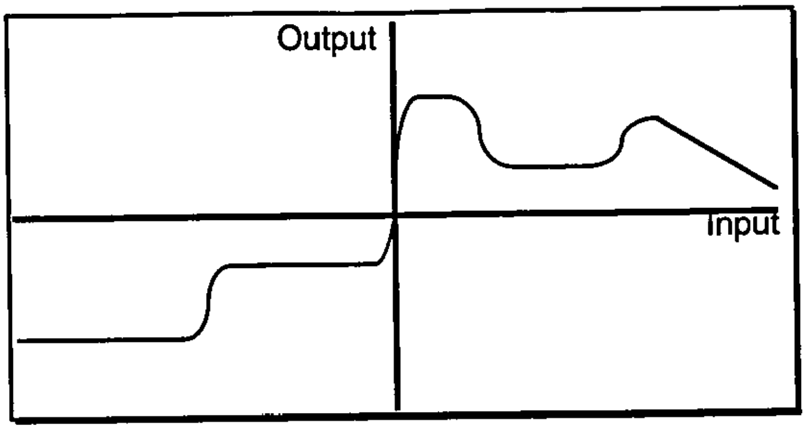


Figure 5.4 The small input tolerance required to give a much looser output tolerance for a continuous network.

The very tight tolerances required to give reasonable answers suggest that the continuous model will be harder to train. There are many other problems with this type of network, including an even greater lack of tools for analysing the hypotheses described earlier in this chapter.



.6 The approximation used to model the function in figure 5.5

he approximation shown in figure 5. would produce the correct answer over the domain there are regions around the points of discontinuity where the error ome large. If a tight tolerance is required then the normal feed forward networks be used. These represent another class of impossible functions.

in size

forward neural network is capable of calculating an output for every possible Therefore there are problems for which a network will deliver a real answer o real answer exists. Given that no real answer exists, the network must be ng outside of the required tolerance. An example of such a function is

$$y = \sqrt{9 - x^2}$$

uation represents a circle of radius 3, centred at the origin. A neural network return a value for y given that x = 100. No real answer should exist in this case, nting yet another class of problems for which feed forward neural networks be used in the classical manner.

STOR-A-FILE IMAGING LTD

DOCUMENTS OF POOR ORIGINAL HARD COPY

networks. The problem is to map a continuous function f that has inputs, I, and returns outputs, O, in a Boolean space. There is a predicate, P, denoting that a particular instantiation of I and O are consistent with each other. By constructing a network to model the predicate P, with both I and O as inputs, it is possible to reproduce the original function f in a $R \Rightarrow B$ network.

The simple ball throwing example given earlier would require a two input network. These would be the original input, angle of throw, and a new input, distance to first bounce. The networks output would classify the inputs as consistent with each other or as an impossible combination. The hypothesis derived from such a network is shown below. The dark region show possible combinations.

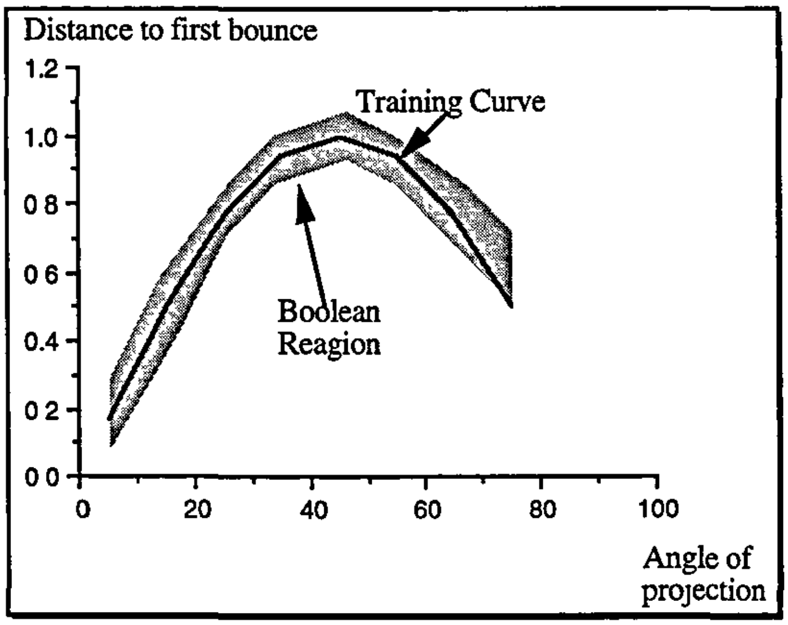


Figure 5.10 The hypothesis formed by a Boolean network when trying to model the data in figure 5.2.

Training Problems

The method as described above works with small tutorial examples. However, it does not extend well to bigger problems. Consider the example continuous function shown in figure 5.11. The positive examples (shown as heavy crosses) are the original training

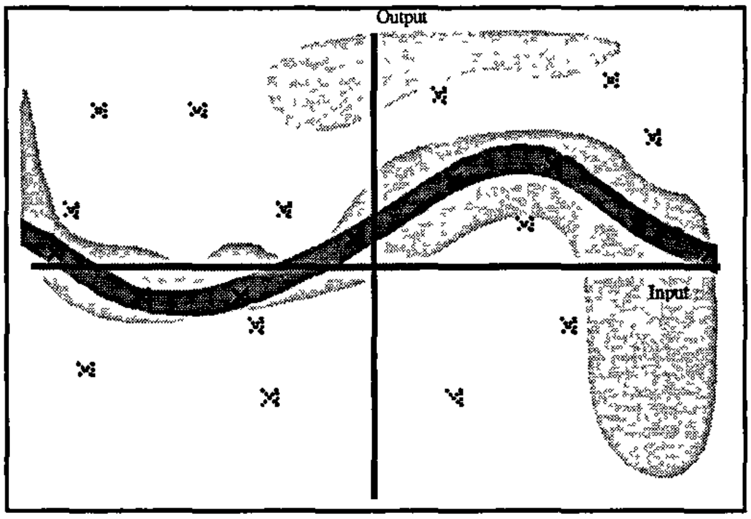


Figure 5.12 The lightly shaded region is the area that is learnt from the training points shown. Although this is a correct solution to the training problem it does not correctly model the continuous function.

Producing an even covering of training points and limiting the width of the model produces a much larger training set. An example of the number and position of a correct set can be seen in figure 5.13.

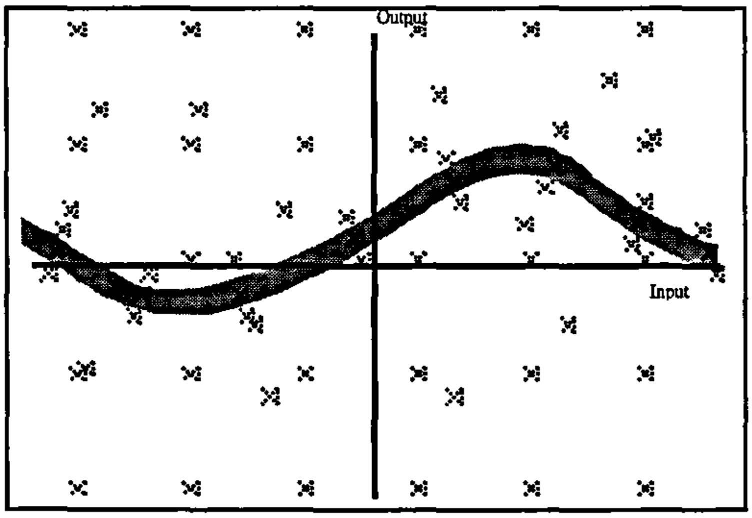


Figure 5.13 The training points required to produce a correct Boolean model of the continuous function.

Although the method of creating a correct training set works, we have introduced a new problem. As the size of the problem increases the Boolean training set can become extremely large, leading to very slow training. Further, the proportion of positive and negative examples will become very biased. In large problems less than one percent of the examples may be positive. Thus, the output neuron will return false irrespective of its inputs. The small proportion of positive examples makes it almost impossible for the network to converge past this easily found local minimum.

Training Solutions

The problems connected with the training and production of examples detailed in the last section prevent raw computing power being used to compute a satisfactory Boolean network. It is necessary to construct a specialist training mechanism capable of producing networks of this type in an acceptable time. The problem of producing a thin accepted region can be split into two parts (figure 5.14). The answers are then combined later using a single neuron that computes a simple Boolean AND function. This mechanism produces a much larger percentage of positive examples in both sub problems, removing the difficult local minimum of Output = False.

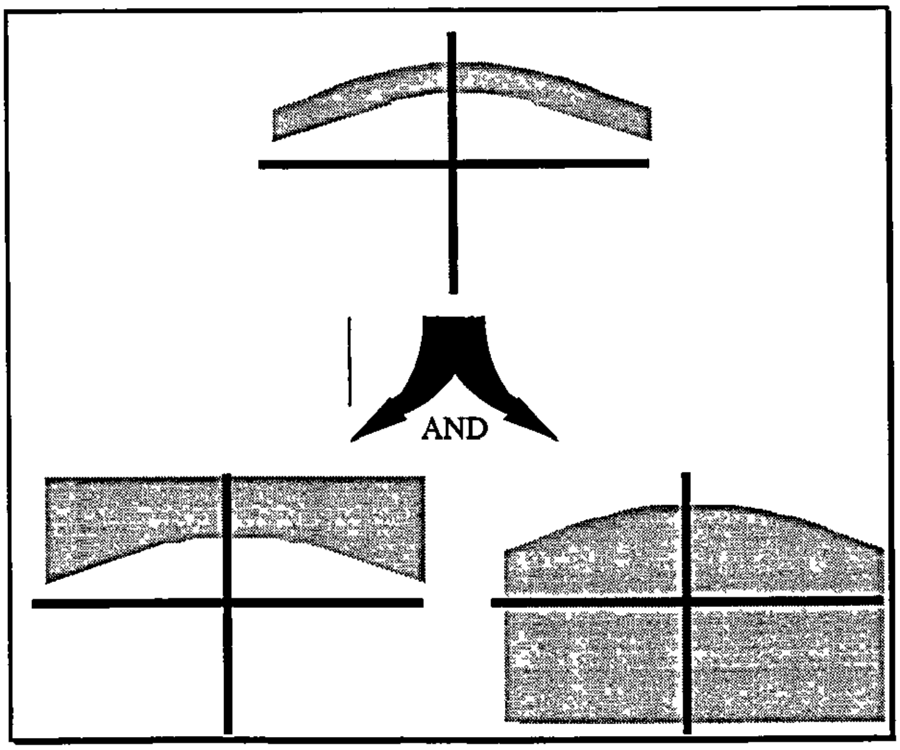


Figure 5.14 Showing how a continuous model can be split into two sub problems.

them a better solution. Justification for their use can be gained by considering the classes of problems that continuous output neural networks cannot or have difficulty in modelling.

Single verses multi-valued functions -- The solution

The example of a multi-valued function given earlier was

$$y = \sqrt{x}$$

This could not be modelled because a feed forward neural network can only perform a one to one mapping. Therefore it is possible only to build a network to return either the positive or negative answers, but not both. Once the real valued output becomes part of the input space it is possible to map both of the possible answers to give a true response. This effectively models a multi-valued function.

Continuous and dis-continuous functions

Discontinuous functions could not be modelled using the traditional technique because of the limitation imposed by the learning algorithm. Boolean functions, by comparison, are very good at implementing discontinuous functions. A simple example is three input parity; this function has two discontinuous regions that require the output to fire.

If a Boolean classification network can produce discrete regions then different regions can be used to represent different portions of a discontinuous function. The example used earlier to demonstrate the difficulties with these functions could be modelled in five different regions, figure 5.17.

Range size

The Boolean network uses the old output as an input. Neural networks can operate over any input domain, so the limitations of range size on the original function do not apply. An example demonstrates the advantages gained when modelling over a larger range. Here the network has been trained to model the function $y = x^2$ over the domain -3 .. 3. The result is shown in figure 5.18.

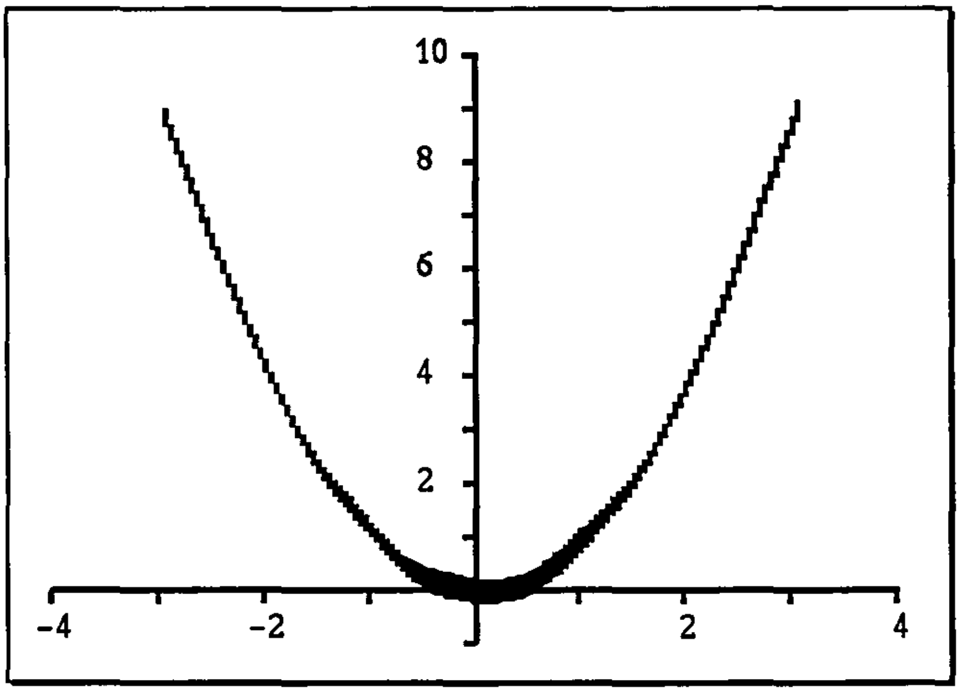


Figure 5.18 The hypothesis resulting from combining the sub problems for $y = x^2$. The maximum error between this and the correct answer is 0.15.

The greatest deviation from the correct answer for any point within the training region given by this network was 0.10385. This compares extremely favourably with the same function trained as a traditional continuous network (figure 5.19). The errors in this example are large because it is necessary to multiply the network's result by 10 to get a large enough range, therefore any small errors in the hypothesis get magnified by 10. The Boolean version does not suffer from this problem.

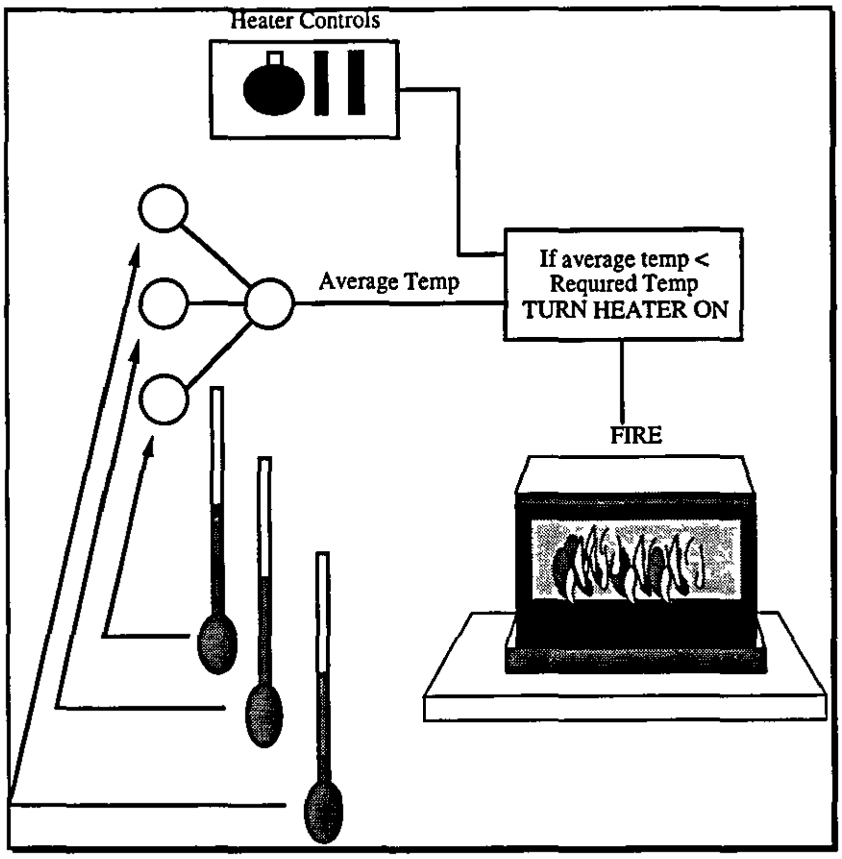


Figure 5.20 A **VERY** simple control system

What is most important is that the model of the continuous function must deliver the real value of the temperature. It is necessary to construct a method of extracting a continuous value from the Boolean models that this chapter conjectures should replace the continuous versions.

Extracting Continuous Output from a Boolean Model

The extraction of the correct real output value from a Boolean model consists of calculating the input value that causes the network to return true. This calculation can be performed by using a modification of the back propagation training rule to change the input values instead of the interconnecting weights.

If this method is used to search for an set of inputs that give the output value of one, then the inputs have been forced into consistent values. Further, if no updates are made to the original inputs and only the old "output" is modified then it is possible to derive a correct "output" for any inputs.

Worked Example

Returning to the colour perception problem from chapter four. The problem is to model perceived values of hue, brightness and colourfulness from the many different parameters that effect the way the human eye performs. This very hard problem has be the subject of research for well over 100 years [Young, Luo, Hunt, Helmholtz]. Each output is a complex continuous function varying between 0 and 1.

A set of 120 training points were used to train networks to model the hue output. Different versions of the network were trained using the continuous and Boolean paradigms. The graph below shows the errors between the network results and the correct answers for a test set of 50 test points.

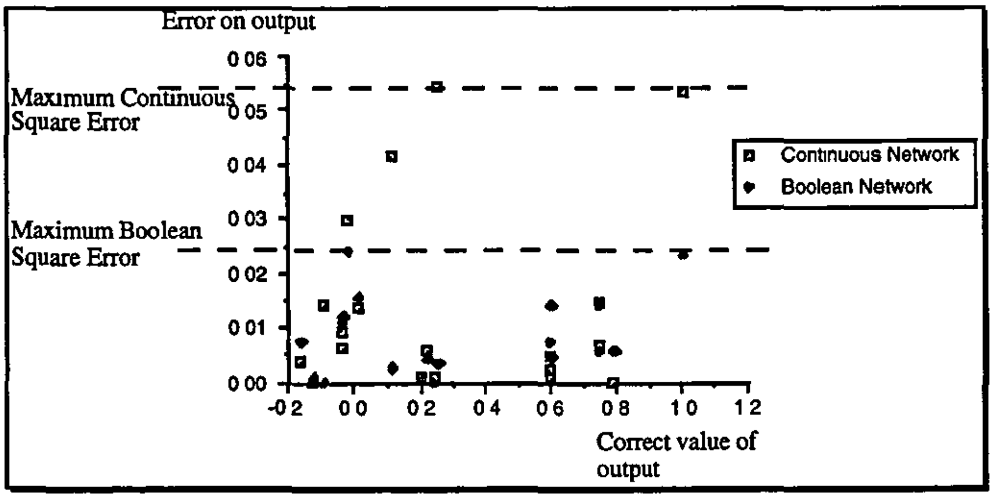


Figure 5.22 A comparison of the accuracy achieved using a Boolean model and a traditional real valued model.

These results were obtained after 30000 training iterations on both models, giving them adequate time to achieve their optimum solutions. As such, both models had reached local minima and further training would produce no change. The results show that the Boolean model is capable of returning a lower average error and, more importantly, a much smaller maximum error. In the example, the traditional network could return the correct answer to a tolerance of ±0.235 compared to the Boolean tolerance of ±0.158. The Boolean network represents a large improvement in performance.

Summary

This chapter has investigated methods of increasing the robustness of neural networks that are modelling continuous functions. These were the largest class of problems that could not be dealt with using the techniques developed in chapters 2 & 3.

There are several problems with traditional models of continuous functions. They are limited to single valued continuous functions over an infinite domain, i.e. they cannot model multi-valued, discontinuous functions or those with smaller domains. Further they have a limited range; the output of the final neuron is limited to -1.0 ... +1.0. If scaling is used to increase the range then accuracy becomes a major problem.

The method of representing a continuous function as a Boolean predicate overcomes all of these problems. This is achieved by modelling a predicate that calculates whether a supplied input/output pair is self consistent. The old output has now become an input to the new network; the network tells us if our guess was correct.

In some circumstances a real valued output is necessary and the Boolean predicate is insufficient. This chapter closes by showing how back propagation can be modified to complete a search for the correct "output" value, thus returning to a continuous model but now without the limitation detailed earlier in the chapter.

CONCLUSION

CHAPTER 6

While neural networks have demonstrated an ability to implement many complex systems, they cannot as yet be relied upon. Until the design and verification systems for neural networks become robust they cannot be widely used in the important industrial, commercial or safety critical environments. This thesis demonstrates that analysis is both possible and useful,. Further it shows that there is a strong link between analysis and design, and demonstrates how the good practice of design can be extended to cover several new classes of problem.

Chapter two attempts to show that there is a strong link between design and verification. Both fields strongly encourage the other, and to a certain extent rely on the other; design is improved if it is possible to check the design works and it is easier to verify networks that are properly designed. The currently available systems for network design are limited in scope. The work has therefore addressed two separate problems

> To demonstrate that is possible, and useful, to analyse trained feed forward networks.
>
> To extend the good practice of design to classes other than Boolean classification feed forward networks.

There have been some previous results aimed at representing classification networks as Boolean type functions. Analysis of one of these shows major representational flaws. While simple Boolean functions can be a very long winded description of a network it is always possible to produce a correct analysis. I feel that the ability to describe any classification network indicates that simple Boolean functions are perhaps the best system to use. A method for extracting a Boolean function from the weights and bias of a neuron makes up one of the major arms of this thesis. The Boolean representations of the neurons can be combined to represent the whole network, demonstrating that is possible to calculate a model of a neural classification system.

The extraction of a Boolean representation for a network is possible, but is it useful? The answer to this question is fairly complex. In a classification network the Boolean representation can be used to build a rule based system. In chapter three the thesis shows how such a system can be simplified and tested using simple image enhancement strategies. This is best illustrated by a simple controller used to control a pole balancing machine (figure 3.17). While the neural controller acted correctly in all of the test cases, an analysis of shows that it cannot be 100% correct because it has an asymmetric response and the problem is symmetric. So, for classification systems, the analysis system appears to be both possible and useful.

Image recognition is one class of classification system for which a rule based approach appears to have little meaning; consider trying to write down the rules for how you recognise somebody's face or how you can 'see' a tree. These problems need a different mechanism for viewing the analysis. The Boolean rules still contain all the information but in an inappropriate form.

The method of interpreting the rules must change depending on the domain of the neural system, but the actual 'rule extraction' idea is appropriate across many fields. Optical character recognition is an example of a classification network for which the derived rules are nearly meaningless to a human observer. A different method of delivering the information to the user is presented, demonstrating that by changing the representation of the rules for different domains, the 'rule extraction' methods of analysis is still useful.

One of the major limitations of rule extraction is the size of the network. As the size grows the Boolean representation becomes increasingly complex. While this limitation is discussed in some detail in chapter three it is also the motivation for the work in chapter four. If the size of a network is the limiting factor then a method of reducing this must be valid goal. Increasing the power of an individual neuron means that fewer of them are required. Chapter four introduces a more powerful neuron and shows how they can be trained using standard back propagation.

It has already been shown that Boolean analysis is useful. Using the more powerful 'general activation neurons' does not effect the representation of the rules. However, the methods used to extract the raw Boolean rules from the neurons must be updated to cope with the added complexity. This represents the next step necessary to further expand the class of problems that can be fully analysed.

The last chapter investigated perhaps the most useful set of networks not covered by the use of Boolean classification systems, namely those that are modelling Real to Real mappings. There are several problems with traditional models of continuous functions. They are limited to single valued continuous functions over an infinite domain, i.e. they cannot model multi-valued, discontinuous functions or those with smaller domains. The final piece of work demonstrated a method of representing a continuous function as a Boolean predicate overcomes all of these problems. The correct design of networks for these problems may make it possible to produce an analysis system, but until this was achieved it is unlikely that analysis would be possible. As a Boolean mapping is used within the new model it may be possible to extend the work contained in the earlier chapters to cover continuous models.

In conclusion this thesis has tried to show that design and analysis for connectionist systems are desirable, and that there were several holes in the available techniques.

Methods have been presented that show that analysis is possible and desirable for classification networks. The biggest limitations are the size of the networks and that they are only applicable to classification systems. A method of reducing the size of classification networks is presented along with a design methodology for non-classification systems. The extension of the analysis tools to cover these new architectures could be a subject of further study.

REFERENCES

Anderson, K., Cook, G.E., and Gabor, K. 1990 Artificial neural networks applied to arc welding process modelling and control. IEEE Transactions on Industrial Applications, **26**, pp 824-830.

Anderson, K., Cook, G.E., Springfield, J.F. and Barnett, R.J. 1991 Applications of artificial neural networks for arc welding. In Intelligent engineering systems through artificial neural networks, Dagli, Kumara and Shin (eds.), New York, pp 717-728.

Bailey, D.L., Tompson, D.M. and Feinstein, J.L. 1988 Options trading using Neural Networks, Transaction from Neuro Nimes

Bryson, A.E. and Ho, Y.C. 1969 Applied Optimal Control. New York

Burke, L.I. and Rangwala, S. 1991 Tool condition monitoring in metal cutting: a neural network approach. Journal of intelligent Manufacturing, **2**, pp 269-280

Cater,J.P. 1987 Successfully using peak learning rates for 10 (and greater) in back propagation networks with the heuristic learning algorithm. IEEE first international conference on Neural networks, **2**, pp 645-651.

Dowsing, R.J., Rayward-Smith, V.J., and Walter, C.D.	1986	A first course in formal logic and its applications in Computer Science, Blackwell Scientific Publications.
Block, H.D.	1970	The Perceptron: A model for Brain Functioning, Reviews of Modern Physics, **34**, pp 123-135
Fahlman, S.E.	1989	Fast-Learning Variations on Back propagation: An Empirical Study, Proceedings of the 1988 Connectionest Models, pp 38-51, eds. Touretzky, Hinton & Sejnowski, Pub. Morgan Kaufmann
Frean, M.	1990	The Upstart Algorithm. A method for construction and training feedforward neural networks. Neural computation, **2**, pp 198 - 209.
Govekar, E., Grabec, I., and Peklenic, J.	1989	Monitoring of a drilling process by a neural network. The 21st CIRP International Seminar on Manufacturing Systems, Stockholm, Sweden.
Grant, E., and Zhang	1989	A neural net approach to supervised learning of pole balancing, Proc. 4th International Symposium on Intelligent Control, 25-27 September, Albany, New York.
Griffith, J.G.	1966	Inaugural Lecture for the Chair of Applied Mathematics, Bedford College, University of London.

Hall, J.W., and Lu, S.C.Y.,	1992	Emulating human process control functions with neural networks. Knowledge-Based Engineering Systems Research Lab. University of Illinois. pp 145-152.
Hebb, D.O.	1944	The Organisation of Behaviour, Wiley, New York
Helmoltz, H.V.,	1924	Handbook of physiological objects, Vol. 2, ed. Southall, J.P.C., Optical society of America, New York.
Hinton, G.E.	1990	Preface to the Special Issue on Connectionist Symbol Processing, Artificial Intelligence **46** pp 1-5
Hopfield, J.J.	1982	Neural Networks and Physical Systems with Emergent Collective Computational Abilities, Proceedings of the National Academy of Sciences, USA, **79**, pp 2554-2558.
Hopfield, J.J., Feinstein , D.I., and Palmer, R.G.	1983	Unlearning Has a Stabilising Effect in Collective Memories, Nature, **304**, pp 158-159.
Horejs and Kufudaki	1993	Neural networks with local distributed parameters Neurocomputing, **5**, pp 211 - 219

Hunt, R.W.G.	1987	A model of colour vision for predicting colour appearance in various viewing conditions, Colour Research Applications, **12**, pp. 297-314.
Hush, D.R., and Salas, J.M.	1988	Improving the learning rate of Back Propagation with the gradient reuse algorithm, IEEE International conference of Neural Networks, **1**, 441-447.
Jalel, N.A., Mirzai, A.R., Leigh, J.R., and Nicholson, H.	1991	Application of neural networks in process control. In Neural network applications, J.G.Taylor (ed.), Springer, pp 101-113.
Kamarthi, S.V., Cohen, G.S., and Kumara, S.R.T.	1991	On-line tool wear monitoring usıng a Kohonen feature map. In intelligent Engineering Systems through Artificial Neural Networks, New York, pp 639-644.
Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P.	1983	Optimisation by Sımulated Annealing, Science **220**, 671-680.
Kohonen, T.	1984	Self-organisation and Associative Memory. Springer-Verlag, Berlin.
Luenberger, D.G.	1986	Linear and Nonlinear Programming, Addison-Wesley.

Luo, M.R., Clarke, A.A., Rhodes, P.A., Schappo, A., Scrivener, S.A.R., and Tait, C.J. 1991 Quantifying colour appearance, part 1, LUTCHI colour appearance data, Colour Research Applications, **16**, pp. 166-179.

Marchand, M., Golea, M., and Rujan, P. 1990 A convergence Theorem for Sequential Learning in Two Layer Perceptrons. Euro Physics Letters,**11**, pp 487-492.

McCulloch, W.S. and Pitts, W. 1943 A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, **5**, pp 115-133.

Messenger, J.B. Nerves, Brains and Behaviour, Studies in Biology, **114**.

Messom, C.H., Hinde, C.J., West, A.A., and Williams, D.J. 1992 Designing neural networks for manufacturing process control systems,Proc. 1992 International Symposium on Intelligent Control, August 1992, I.E.E.E. Publishers.

Messom, C.H. 1992 PhD Thesis, Dept Computer studies, Loughborough University of Technology.

Mezard, M., and Nadal, J.P. 1990 Learning in Feedforward neural networks: The Tiling Algorithm. Journal of Physics A : Maths & General.

Mihalaros, M.N. 1992 MSc Thesis, Dept Computer studies, Loughborough University of Technology.

Minsky, M.L., and Papert, S.A. 1969 Perceptrons: an Introduction to Computational Geometry, MIT Press.

Poppel, G., and Krey, U. 1987 Dynamically Learning Processes for Recognition of Correlated Patterns in Symmetric Spin Glass Models, Europhysics Letters, **4(9)**, 979-985

Rosenblatt, F. 1958 The perceptron: a Probabilistic Model for Information Storage and Organisation in the Brain. Psychological Review, **65**, pp 386-408.

Rosenblatt, F. 1962 Principles of Neurodynamics, Spartan, New York.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. 1986a Learning Internal Representations by Error Propagation, in Rumelhart and McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1: Foundations. MIT press.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. 1986b Learning Representations by Back-propagating Errors, Nature **323**, pp 533-536.

Rumelhart, D.E. and McClelland, J.L. 1986 On learning past tenses of English Verbs. PDP vol. 2 MIT press.

Sejnowski, T.J., and Rodenberg, C.R.,	1986	Parallel Networks that Learn to Pronounce English Text, Complex Systems **1**, pp 145-168.
Sietsma, J., and Dow, R.J.F.	1988	Neural net pruning- Why and How. In IEEE International conference on Neural Networks, **1**, pp 665-672.
Solla, S.A., Levin, E., and Fleisher, M.	1988	Accelerated learning in Layered neural networks. Complex systems, **2**, pp 625-639.
Tanni, J.„ Hirobe, K., Niida, K., Koshijima, I., and Murakamı, H.	1989	New learning algorithm for rule extraction by neural network and its applıcation. Proceedings of American Association for Artificial Intellıgence.
Venkatasubramanian, V., and Chan, K.	1989	A neural network methodology for process fault diagnosis, AIChE Journal, December 1989.
Wang, J., and Malakooti, B.	1992	A feed forward neural network for multiple criteria decision making. Computers and Operatıons Research, **19**, pp 151-166
Watrous, R.L.	1987	Learning Algorithms for Connectionest Networks: Applied Gradıent Methods of Nonlinear Optimisation. IEEE Conference on Neural Networks, San Diego, CA, USA. **2**, pp 619-627
Young, T.	1802	On the theory of light and colours, Royal Phılosophical Society, **92 (12)**.

Appendix A

The following papers, containing results from this thesis, have either been published or have been accepted for publication.

Interpretation of Neural networks as Boolean transfer functions

Knowledge-Based Systems

Volume 7, Number 3, September 1994

pp 207 - 213

Interpretation of neural networks as Boolean transfer functions

G P Fletcher and C J Hinde

An algorithm for converting neural networks into Boolean functions is presented. The absence of such an algorithm has been identified in the literature as a significant problem, and the solution shown in the paper is both complete and efficient. The analysis of the algorithm shows it to have a time complexity of better than $2^{N-1} - 2^{(N/2)-1} + 1$.

Keywords: neural networks, rule extraction, rule induction

In order to achieve the full potential of artificial intelligence it is necessary to use results from both symbolic and connectionist work. Both offer different views and neither can be expected to encompass the full power of the other[1]. There have been many reports over the past few years on methods of training neural networks to emulate different rule based systems[2,3]. The two methodologies are seen to be complementary and can be combined into hybrid systems[4,5] The modelling of neural networks using Boolean algebra is the natural obverse of earlier results allowing Boolean rules to be expressed as a network[6], and it is a central issue in connectionist research[7,8]. Venkatsubramanian[7] in 1989 suggested that 'the combination of neural-based and knowledge based systems may improve the performance speed and reduce the complexity and time required for building intelligent systems'. Zhang *et al.*[8] stated more recently in 1992 that 'the combined use of knowledge based expert systems techniques with the neural-network technique could enhance the advantages in both areas and would be a useful future research topic'. Hinton[4] in 1990 expresses the view that 'the problem is to devise effective ways of representing complex structures in connectionist networks without sacrificing the ability to learn the representations My own view is that connectionists are a very long way from solving this problem'.

Mathematical proof of the properties of neural networks would allow them to be introduced into safety critical environments where, at the moment, all work must be of a symbolic nature; this addresses the trust that can reasonably be placed in trained neural networks[9]. The Boolean derivation of the network could then be used to check that the basic architecture is correct and that the weights selected are of the correct order of magnitude and sign. If mistakes are found then the result can be used to help select new training examples, speeding up the learning cycle by preventing useless examples being presented.

BOOLEAN REPRESENTATION OF NETWORK

Neural networks are able to accept continuous valued inputs. Thresholding beyond the first layer means that all transformations beyond the first are Boolean transformations, whereas the first layer may be regarded as a quantisation layer splitting the input space into regions which are then combined logically by the subsequent layers. In this paper we only address the transformations of the second and subsequent layers; the first layer may be interpreted as the result of relational operators comparing real valued inputs. The use of a sigmoid function changes this view of thresholding somewhat, but with a fully trained network the sigmoid function will behave as a very close approximation to a threshold function.

BOOLEANS THROUGH TRUTH TABLES

It is possible to produce a correct Boolean representation by applying every input pattern and producing a truth table. This truth table can then be turned into a Boolean

Department of Computer Studies, University of Technology, Loughborough, Leics. LE11 3TU, UK
Paper received 4 March 1993 Revised paper received 28 May 1993 Accepted 19 July 1993

function using an algorithmic implementation of Karnaugh maps[10]. Both stages of this method have a time complexity of $O(2^n)$, giving a minimum total time complexity of 2^{n+1}. This time complexity is prohibitively large and prevents this approach being used on any reasonably sized network.

O OPERATORS

Initial investigations into this problem used a piecemeal approach, splitting off the Boolean operators one by one. This produced interesting results for small test data sets using two input nodes but failed to generalize properly to larger systems. The intention is to transform each neuron into a Boolean function, back substitute, and then apply a simplification procedure to the resulting collection of rule sets to derive the overall transformation function that the net represents Although a thresholded neuron beyond the first layer can always be represented as a Boolean function, the form of this function cannot be restricted to a simple representation where each input is mentioned only once: n input AND and OR are examples of such simple functions. The interest in such functions stems from the problems of scalability which become increasingly relevant as the number of inputs rises and the search space for the correct Boolean function grows exponentially. If a set of operators can be found that represents any neuron with this restriction then an effective algorithm could be discovered and a concise representation would be found. A class of operators which we refer to as the O and A operators, as they are generalizations of OR and AND, has consequently been investigated and presented[11], where

$$O^n_m(I_1, \ldots, I_m)$$

is true if at least n of the m inputs are true.

For example, a three input 'or' function becomes

$$O^1_3(I_1, I_2, I_3)$$

and this can be represented as the simple Boolean function $I_1 \vee I_2 \vee I_3$

$$O^2_3(I_1, I_2, I_3)$$

is represented as the simple Boolean function $(I_1 \wedge I_2) \vee (I_2 \wedge I_3) \vee (I_1 \wedge I_3)$.

$$O^3_3(I_1, I_2, I_3)$$

is represented as the simple Boolean function $(I_1 \wedge I_2 \wedge I_3) \vee (I_1 \wedge I_2 \wedge I_3) \vee (I_1 \wedge I_2 \wedge I_3)$, or, more simply, as $(I_1 \wedge I_2 \wedge I_3)$.

The A operators are effectively the complementary function based on AND.

$$A^n_m(I_1, \ldots, I_m)$$

is true if at least one element in each combination of n out of the m inputs is true. For example

$$A^1_3(I_1, I_2, I_3)$$

is represented as the Boolean function $(I_1 \wedge I_2 \wedge I_3)$.

$$A^2_3(I_1, I_2, I_3)$$

is represented as the Boolean function $(I_1 \vee I_2) \wedge (I_2 \vee I_3) \wedge (I_1 \vee I_3)$.

Both these operators can be represented by a single neuron demonstrating that a neuron is more complex than a simple n input AND or OR. They are complete up to three inputs and therefore provide a means of representing all possible neurons with an input dimension of three or less. Perhaps more significantly, and what encouraged further work, is that all n-input parity neural networks can be implemented by O operators[12]. The work presented by Mihalaros[11] presented no complete analysis of how to find the O operator that matched a general neuron even if all the possible neurons in higher dimensions were covered.

MATCHING NEURON TO O OPERATOR

The possible functions that can be represented by a single neuron fall into distinct groups. Each of these groups will contain exactly one natural neuron and possibly some real ones, where we define 'natural' and 'real' as follows. *Natural* means that all the weights and the bias of the neuron are positive. *Real* means that at least one of the weights or the bias of the neuron is negative.

The O operator representations of all the neurons in a group are similar, the only difference being that, for a real neuron, some of the input variables will be negated. Two statements can be made about the connection between the O operators and natural neurons.

- *O operators that do not contain negated inputs represent natural neurons:* Increasing the number of inputs set to true can only cause the output to stay static or change from false to true. This is exactly the same behaviour as that for a natural neuron. Natural neurons monotonically become 'more true' as more inputs are set to true.
- *O operators that contain negated inputs represent real neurons* There must be a set of the inputs set to true such that there is a negative input which when set to true will cause the output to change from true to false. This situation cannot be represented by a natural neuron and so represents a real neuron Real neurons do not monotonically become 'more true' as more inputs are set to true.

One way to match a real neuron to an O operator is t convert the neuron into a natural neuron, match this ne

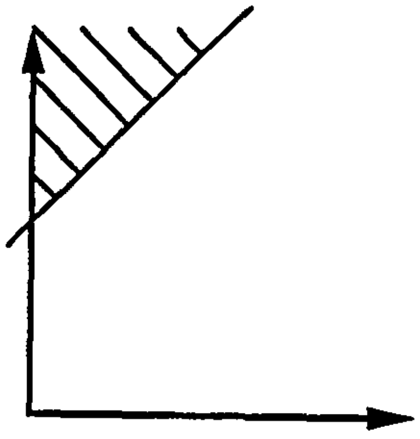


Figure 1 Two dimensional hyperplane with one negative and one positive weight

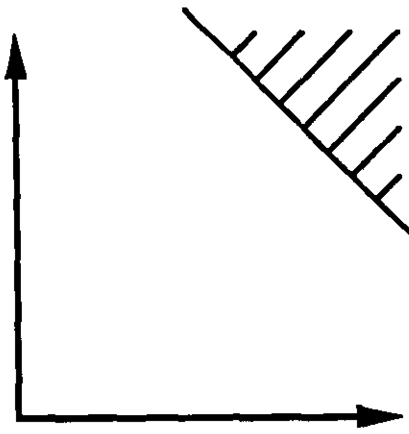


Figure 2 Transformed hyperplane with all weights positive

problem to an O operator, and then convert the O operator back to match the original by negating some of the inputs

CONVERTING REAL NEURON INTO NATURAL NEURON

The crosshatched region in *Figure 1* is represented by the equation

$$I_2 - I_1 > 0.5$$

To change the negative weight from I_1, it is necessary to move the origin to position (1,0). The new equation for the graph is therefore

$$I_2 + I_3 > 1.5$$

where

$$I_3 = \sim I_1$$

This gives the new diagram shown in *Figure 2*. If an O operator F is found for this new neuron then all instances of I_3 in F should be replaced by $\sim I_1$.

INPUTS	I1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
	I2	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
	I3	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
	I4	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
OUTPUTS		0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1

Figure 3 Arbitrary four dimensional function
[The function is linearly separable and so can be implemented by a single neuron]

Any inequality can be treated in this manner using the following algorithm Let the inequality be

$$w_1 I_1 + w_2 I_2 + \ldots w_d I_d > B$$

Let the new inequality be

$$W_1 I_1 + W_2 I_2 + \ldots W_d I_d > D$$

For all n in $1 \ldots d$,

$$W_n = |w_n|$$

and

$$D = B - \sum \mathrm{f}(w_n)$$

where

$$\mathrm{f}(X) = \begin{cases} 0 & X > 0 \\ X & X \leqslant 0 \end{cases}$$

MATCHING NATURAL NEURONS TO O OPERATORS

If the function represented by the natural neuron were to be drawn onto a layered graph that represents the possible Boolean input space, where graph nodes are connected if they differ by only one input and graph nodes are in the same layer if they have the same number of inputs set to true, then only graph nodes that have no children need to be represented; we are only interested in the infimums. The four dimensional function shown in *Figure 3* would be represented as the lattice shown in *Figure 4*.

Figure 5 shows the set of infimums taken from the lattice shown in *Figure 4*, these are the only nodes that need to be represented. This is because in an 'at least function', all the others are automatically covered, as the tinted nodes are the 'least' elements. Finding the O operator for these nodes on the graph can be done using the following algorithm.

- *Step 1:* If only a single node is marked, then the O operator is

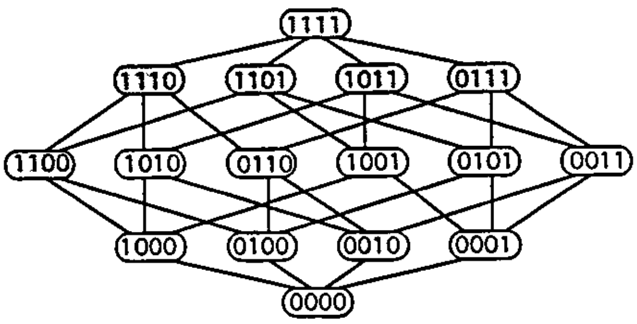


Figure 4 Four dimensional function tabulated in *Figure 3* shown as lattice
[The tinted nodes correspond to an output of 1]

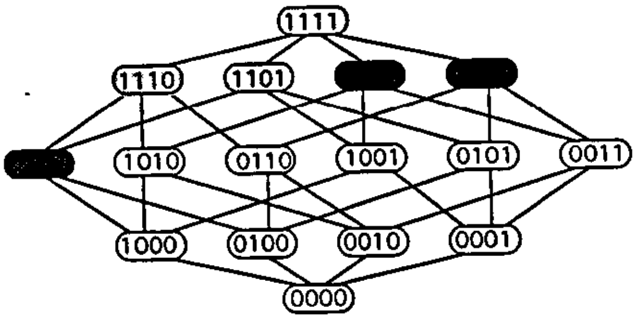


Figure 5 Infimums
[The more heavily tinted nodes represent the infimums of the tinted nodes in *Figure 4*]

$O^n_n(I_1 \ldots I_n)$

where $I_1 \ldots I_n$ are the dimensions that are set to 1 in the required node For example, if the required node is (1,0,1,1), then the O operator is

$O^3_3(I_1, I_3, I_4)$

- *Step 2* Where there are Z source nodes, $Z > 1$, calculate the O operator for each source node ignoring the others; group these together. To find a single O operator that represents all other O operators in a group, do the following:
  - *Step 2 1.* If there is only one O operator in the group, then that O operator is the answer.
  - *Step 2.2.* If the O operators in a group all form a rotary set then they can be expressed as a single O operator. That is when every O operator is of the form

    $O^n_n(I_a \ldots I_b)$

    where every O operator has the same value of n, and there is a set of operands $I = \{I_1 \ldots I_m\}$ such that the input operands $I_a \ldots I_b$ of every O operator are members of I, and, for every combination C of n elements from I, there is an O operator $O^n_n(C)$ in the original group Then the answer is $O^n_m(I)$.
  - *Step 2.3:* If the intersection of all the operands of the different O operators in the group is non-empty then calculate this intersection and call this set K_1. Remove this set from the original O operators to form new groups of O operators S If K_1 is not empty then the answer is $O^n_n(K_1, P(S))$, where $P(S)$ is the single O operator that is the simplification of the group S. n is the length $(K_1) + 1$ Apply the algorithm to S to derive $P(S)$.
  - *Step 2.4:* If the group can be split into subgroups such that members of subgroups share common operands with each other and not with members of any other subgroup then do this. For example, the O operators $O^n_m(I_1, I_2)$, $O^n_m(I_2, I_3)$ and $O^n_m(I_3, I_9)$ are all in the same group, and the O operator $O^n_m(I_4, I_5)$ is in a different subgroup. For each subgroup find the O operator that is the simplification of all the members, by starting the algorithm again on the subgroup. The answer is $O^1_n(O_1 \ldots O_n)$, where there are n subgroups and O_k is the O operator that is the simplification of the Kth subgroup.
  - *Step 2.5*: If the group does not match with any one of the first four cases then there is no single O operator to represent all the members

Example 1

Calculate the O operator for the four dimensional function with the infimum nodes (1,1,0,0) and (1,0,1,1). The O operators are therefore $O^2_2(I_1, I_2)$ and $O^3_3(I_1, I_3, I_4)$

- *Step 1* Apply the third rule. The answer is $O^2_2(I_1, P)$, where P is the simplification of the group containing the O operators $O^1_1(I_2)$ and $O^2_2(I_3, I_4)$
- *Step 2·* Apply the fourth rule to P. Each element of the group contains different operands and is therefore in a separate group. The answer is $O^1_2(R, S)$, where R is the simplification of the subgroup $[O^1_1(I_2)]$ and S is the simplification of the subgroup $[O^2_2(I_3, I_4)$.
- *Step 3·* Apply the first rule to R and S. Now no more simplification is necessary, and the final answer is $O^2_2(I_1, O^1_2(O^1_1(I_2), O^2_2(I_3, I_4)))$

Example 2

Calculate the O operator for the four dimensional function shown in *Figure 6*

The three source nodes are (1,1,1,0), (1,1,0,1) and (1,0,1,1) Therefore the starting group contains the three O operators $O^3_3(I_1, I_2, I_3)$, $O^3_3(I_1, I_2, I_4)$ and $O^3_3(I_1, I_3, I_4)$.

- *Step 1* Apply the third rule. The answer is $O^2_2(I_1, P)$, where P is the simplification of the group containing the O operators $O^2_2(I_2, I_3)$, $O^2_2(I_2, I_4)$ and $O^2_2(I_3, I_4)$.

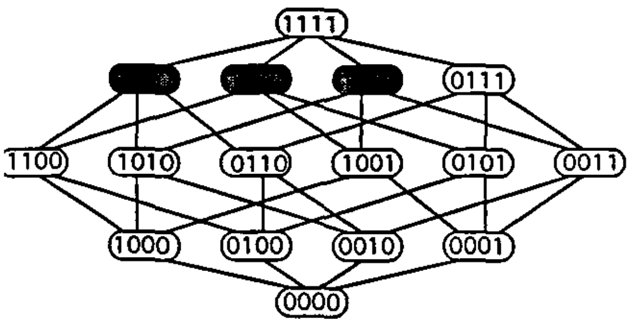


Figure 6 Four dimensional function

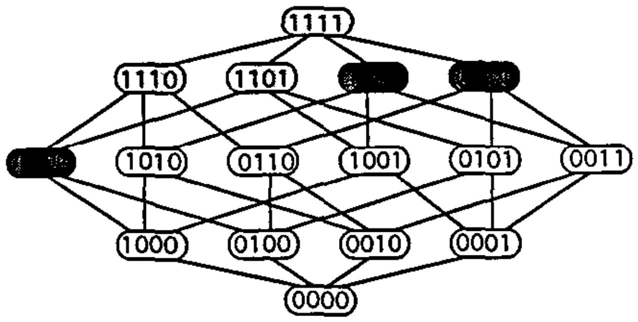


Figure 7 Four dimensional function that cannot be represented by O operators

- *Step 2* Apply the second rule to P. This gives an answer of $O^2_3(I_2, I_3, I_4)$ Now no more simplification is necessary, and the final answer is $O^2_2(I_1, O^2_3(I_2, I_3, I_4))$

WHERE O OPERATORS BREAK DOWN

For the function represented by the graph shown in *Figure 7*, the source nodes are (1,1,0,0), (1,0,1,1) and (0,1,1,1) Therefore the initial group contains the O operators $O^2_2(I_1, I_2)$, $O^3_3(I_1, I_3, I_4)$ and $O^3_3(I_2, I_3, I_4)$. None of the first four rules can be applied for the following reasons: (a) for the first rule, there is more than one O operator in the group, (b) for the second rule, not all the values of n are the same, (c) for the third rule, no operand is common to every member of the group, and (d) for the fourth rule, the group cannot be split, as every member would be in the same subgroup.

According to the fifth rule, the function cannot be represented. An exhaustive search provides an alternative verification of this. As the function is linearly separable, it can be represented using a neuron. Therefore O operators are incomplete, and this is similarly true for A operators. Unfortunately the incomplete elements of the O operator set are not representable using A operators.

CONCLUSION ON USEFULNESS OF O OPERATORS

How to find the O operator, if it exists, for any neuron of a neural network has been shown This has been achieved by finding a similar problem that can be solved and converting this back to provide a solution for the required problem. However we have also shown that O operators are not complete for four or more dimensions, and thus they cannot in general be used to analyse such networks

Where it is possible to use O operators, they provide a concise and clear representation, and thus they are ideal for large networks with low connectivity

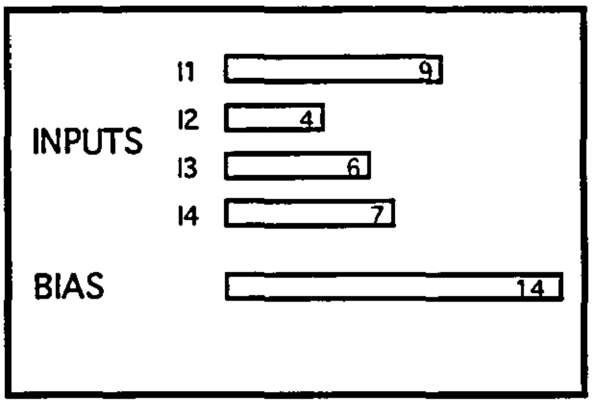


Figure 8 Natural neuron as knapsack problem
[It is necessary to fit the input 'parcels' into the bias 'package']

SOLVING NATURAL NODES CORRECTLY

Natural perceptrons can be turned into Boolean functions with relatively little effort. It is necessary to find which sets of inputs are the minimum required to turn on the output. (These are the same sets that it was necessary to model using the O operators) This problem is similar to the classic knapsack problem, which has an exponential time complexity[13].

Instead of an attempt being made to find a set of inputs that exactly fit the bias, the required answer is all the sets of inputs that just exceed the bias. For example, I_1 and I_4 just exceed the bias as $9+7>14$, but if either of the inputs is removed, then the total falls below the required threshold. The complete set for the problem shown in *Figure 8* is I_1 & I_3, I_1 & I_4 and I_2 & I_3 & I_4.

This represents the required Boolean function and can be transformed easily to the conventional Boolean format (I_1 & I_3) or (I_1 & I_4) or (I_2 & I_3 & I_4)

An algorithm to produce one of the parts of the result is given as a pseudocode procedure below. By backtracking, the procedure will yield each part in turn until the whole function has been produced

```
knapsack(Array_of_inputs,Bias,Answer)
  if Bias $\geqslant 0$ then
    for each element $X$ in Array_of_inputs
      New_Array = all elements after $X$ in
                  Array_of_inputs.
      knapsack(New_Array,
               (Bias - $X$),Sub_Answer)
```

```
        Answer = append($X$,Sub_Answer)
    else
        Answer = [ ]
end
```

COMPLEXITY ANALYSIS

Let $C(X)$ be a measure of complexity that is proportional to the time taken to calculate the answers using the algorithm given above. To produce an answer when there is one weight requires one call to the algorithm:

$$C(1) = 1$$

To produce an answer when there are two weights requires two calls to the algorithm:

$$C(2) = 2$$

To produce an answer when there are N weights requires P calls to the algorithm·

$$C(N) = P$$
$$C(N) = 1 + \sum_{A=1\ N-1} C(A)$$
$$C(N) = 2^{N-1}$$

Therefore the worst-case time complexity for the solution of a natural neuron is $O(2^N)$. This is still exponential, but unlike the exhaustive search method described earlier this can be reduced further, or left with a saving in complexity of approximately four times.

REDUCTION OF COMPLEXITY

The algorithm will only start to approach the maximum complexity when most of the weights are required to make the perceptron fire This means an improvement can be made in the solution time if the hyperplane is more than halfway from the origin. The way to do this is to view the problem from the opposite corner of the unit hypercube and solve the opposite problem. This will bring the hyperplane less than halfway from the new origin. The procedure will then need fewer recursive loops to find its answers. For example, the neuron in *Figure 9* (let us call it P) represents a hyperplane that is more than halfway from the origin. Let G be the second neuron such that

$$G = \sim P$$

and then rotate the axis so that G is natural (see *Figure 10*)

This hyperplane is less than halfway to the opposite corner. Once the Boolean function for G, $B(G)$, has been calculated, the Boolean function for P is $\sim B(G)$.

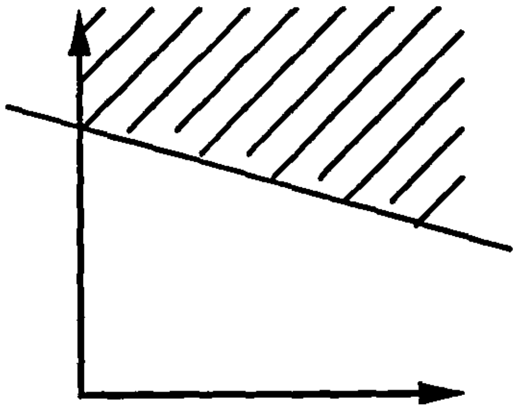


Figure 9 Example of hyperplane implemented by neuron requiring rotation

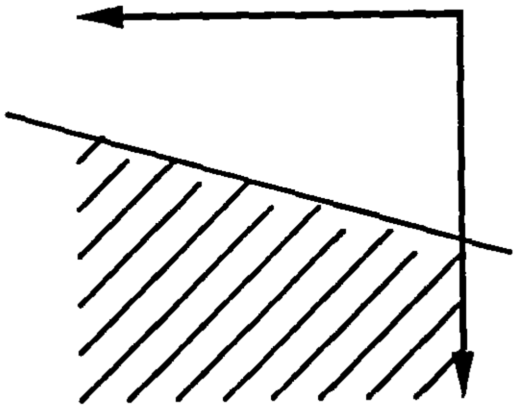


Figure 10 Hyperplane shown in *Figure 9* after rotation

Using this method it will never be necessary to use more than half of the weights to find any part of the solution This changes the values of $C(N)$, as only half of the weights are required, to

$$C(N) = 1 + \sum_{A=N-1\ N/2} C(A)$$

let

$$D(N) = 1 + \sum_{A=1\ldots N-1} D(A)$$
$$D(1) = 1$$
$$D(N) = 2^{N-1}$$

Therefore,

$$C(N) = D(N) - D(N//2) + 1$$
$$C(N) = 2^{N-1} - 2^{(N/2)-1} + 1$$

WHEN TO SIMPLIFY BY ROTATION

It has been stated that the problem should be rotated if the hyperplane is more than halfway to the opposite corner of the hypercube. The problem of deciding what constitutes being more than halfway from the origin is not simple. The obvious, and best, method would be to decide if more than half of the volume of the hypercube is cut off from the origin by the hyperplane. This can seem a very complex task, but analysis shows that, if the boundary is flat, which it must be as it is a single hyperplane, then the side of the hyperplane with the most volume is the side that contains the point exactly in the centre of the cube. Therefore, to test if rotation is necessary, it is only necessary to test to see if the point halfway from the origin is above or below the hyperplane. If it is above then no rotation is necessary, otherwise, rotate. To perform this test, add all the weights together and divide by two. If the total is above the bias then do not rotate, i e. if $(\Sigma \text{weight}_i)/2 <$ bias, then rotate.

FINDING THE MINIMAL ANSWER

The algorithm so far described will return a correct solution. For the purposes of a useful system, the returned answer must not only be correct but also minimal. This can be achieved by sorting the weights into decreasing size before entering them (see the proof). A further advantage of this is that sorting is a good heuristic for reducing the average processing time to find the solution (see *Figure 11*).

Let the input to the algorithm be the weights

$$W_1, W_2, W_3, \ldots, W_n$$

and the bias value B. The mistakes that cause the answer to be nonminimal occur when

$$W_1 + W_2 + W_3 + \ldots + W_m > B \qquad (1)$$

$$W_2 + W_3 + \ldots + W_m > B \qquad (2)$$

Because the algorithm stops as soon as the bias is achieved,

$$W_1 + W_2 + W_3 + \ldots + W_m - W_m < B \qquad (3)$$

However, if the weights are in sorted order, then

$$W_1 > W_m \qquad (4)$$

Substituting Equation 4 into Equation 3 gives

$$W_1 + W_2 + W_3 + \ldots + W_m - W_1 < B$$
$$W_2 + W_3 + \ldots + W_m < B$$

which contradicts Equation 2. Therefore, if the weights are sorted, the two requirements for added complication in the answer cannot occur.

SOLVING REAL NEURONS

Finding the function of a real neuron is a much more complex problem. To solve this, it must be converted into a natural neuron that gives the same results over the Boolean input range.

This is the same idea as was described in deriving the O operator for a real neuron. The answer is similar to the answer for the new question (the natural neuron derived from the real problem), but some of its variables are negated. The variables to be negated are the ones that refer to inputs with negative weights. If the bias is negative, then the problem can be converted to an identical problem, but with a positive bias, by multiplying all values by -1.

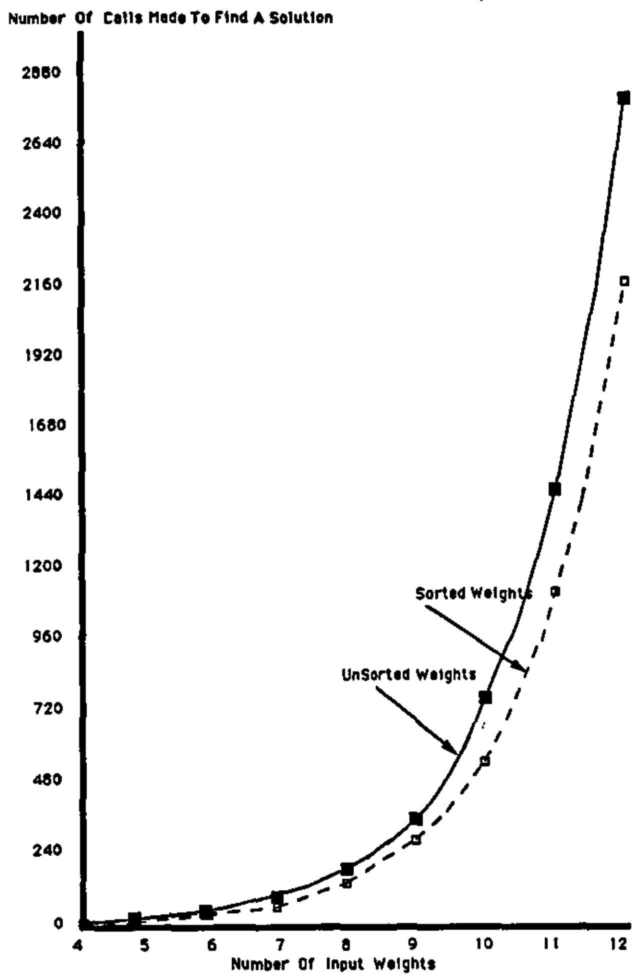


Figure 11 Comparison of speed of execution with sorted and unsorted weights
[Although the lines appear to be close together, the steepness of the curve masks a decrease in calculation time of 30% for the 12 input problem. The graph also shows the impracticability of trying to analyse neurons with many more than 12 inputs]

EXAMPLES

Consider the neuron in *Figure 12* representing a neural network implementation of a two input OR gate.

This neuron implements an OR function. Applying the algorithm to find the minimum inputs to fire this neuron returns two answers A and B which are converted into the correct Boolean function $A \vee B$.

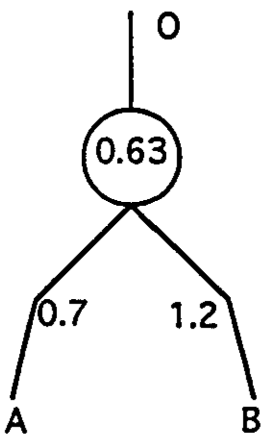


Figure 12 Two input OR gate implemented as neural network

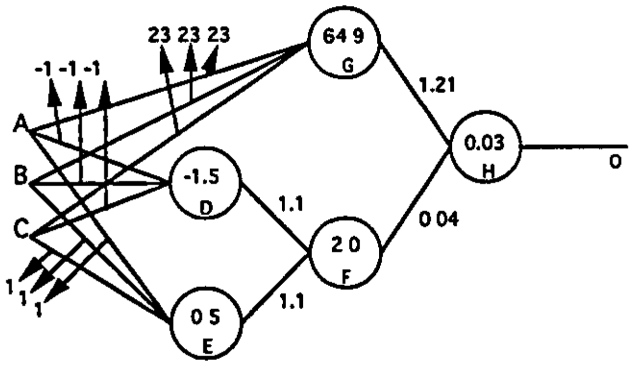


Figure 13 Implementation of three input parity as neural network

A more complex example is the case of three dimensional parity A neural network to calculate this function is shown in *Figure 13*. The Boolean function for each neuron is calculated separately:

$$O = H$$

$$H = G \vee F$$

$$G = \sim(\sim A \vee \sim B \vee \sim C)$$

$$F = \sim(\sim D \vee \sim E)$$

$$D = \sim(A \vee B) \vee \sim(A \vee C) \vee \sim(B \vee C)$$

$$E = A \vee B \vee C$$

Combining these gives

$$O = \sim(\sim A \vee \sim B \vee \sim C) \vee \sim(\sim(\sim(A \,\&\, B) \vee \sim(A \,\&\, C) \vee \sim(B \,\&\, C)) \vee \sim(A \vee B \vee C))$$

which is the correct function expressed in terms of NOR gates.

Once the neural network has been transformed to an equivalent Boolean function, or set of functions, there are many ways of expressing this function. We have chosen to leave this exercise to the reader, as, although it is nontrivial, it is well documented.

CONCLUSIONS

An algorithm for converting neural networks into Boolean functions has been given The absence of such an algorithm has been identified in the literature as a significant problem and the solution shown in this paper is both complete and efficient. Previous attempts, described in the paper, were either incomplete or inefficient. In particular the exploration of generalised Boolean operators proved incomplete above three dimensions and a truth table analysis is inefficient for any reasonably sized neuron. The development of the final algorithm proceeds from converting a 'natural' neuron into a Boolean function to generalising this to 'real' neurons. The analysis of the algorithm shows it to have a time complexity of better than $2^{N-1} - 2^{(N/2)-1} + 1$ if the weight vector is sorted.

REFERENCES

1 Smolensky, P 'On the proper treatment of connectionism' *Behavioural & Brain Sci* 11 (1988) pp 1–23

2 Grant, E and Zhang, B 'A neural net approach to supervised learning of pole balancing' *Proc 4th International Symposium on Intelligent Control* Albany, NY, USA (25–27 Sep 1989)

3 Donne, J D and Ozgunner, U 'A comparative study of neural vs conventional methods for modelling and prediction' *Proc. 1992 International Symposium on Intelligent Control* IEEE (Aug 1992)

4 Hinton, G E 'Preface' *Artificial Intelligence* 46 (1990) pp 1–5 (special issue on connectionist symbol processing)

5 Messom, C H, Hinde, C J, West, A A and Williams, D J 'Designing neural networks for manufacturing process control systems' *Proc 1992 International Symposium on Intelligent Control* IEEE (Aug 1992)

6 Hinde, C J 'A comparative review of neural nets and rule based systems' *Technical Report LUT TR 575* Department of Computer Studies, Loughborough University, UK (1990)

7 Venkatsubramanian, V and Chan, K 'A neural network methodology for process fault diagnosis' *AIChE Journal* 35 12 (1989) pp 1993–2002

8 Zhang, J and Roberts, P D 'On-line process fault diagnosis using neural network techniques' *Trans Inst Measurement and Control* 14 4 (1992) pp 179–188

9 Baum, E B and Haussler 'What size net gives valid generalization?' *Neural Comput* 1 (1989) pp 151–160

10 Dowsing, R D, Rayward-Smith, V J and Walter, C D *A First Course in Formal Logic and its Applications in Computer Science* Blackwell Scientific Publications (1986)

11 Mihalaros, M 'Studying the interpretation of feedforward neural networks using logical functions' *MSc Thesis* Department of Computer Studies, Loughborough University, UK (1992)

12 Messom, C H 'Engineering reliable neural network systems' *PhD Thesis* Department of Computer Studies, Loughborough University, UK (1992)

13 Garey, M R and Johnson, D S *Computers and Intractability* W H Freeman, USA (1979)

Neural Networks as a Paradigm for Knowledge Elicitation

Proceedings of the International Conference on
Artificial Neural Networks, 1994

Maria Marinaro and Pietro Morasso (Eds.)
Springer-Verlag (Pub.)
pp 207 - 213

Neural Networks as a Paradigm for Knowledge Elicitation.

G.P.Fletcher & C.J.Hinde
Dept. Computer Studies
A.A.West & D.J.Williams
Dept. Manufacturing Engineering
University of Technology
Loughborough.

1 Introduction

It has been consistently stated that the hardest part of constructing any knowledge based system is extracting the information from the "expert" (Hart 1986). Rule based systems are one genre of expert systems, in this paper we demonstrate a method for converting a neural network into a set of rules. This maintains the basic advantages of neural networks but redresses some of the disadvantages.

2 The Gluing Machine

The first real application (Chandraker et al. 1990) is the dispensing of adhesive in the manufacture of "mixed technology" P.C.B.s in which through hole and surface mount components are present on the same board. The surface mount components are secured to the board, prior to a wave soldering operation, by a small amount of adhesive. The amount of adhesive dispensed is critically dependent upon several process environment variables.

Messom et al. 1992 produced a sparsely connected 7 input, 5 output neural network system for controlling the adhesive dispensing machine. The trained neural network should therefore be equivalent to a set of rules that could have been learnt by a rule induction package.

Step one in the analysis is to examine the first layer of the network and to express this part as a set of inequalities. This step does not simplify the information but does display it much more naturally than a set of weights on a diagram.

```
IF rise_time < 1.25             IF area > 1 0258
THEN rise_timeflag              THEN midnode1
ELSE ¬rise_timeflag             ELSE ¬midnode1
     .                               .
     .                               .
```

The remainder of the network can then be converted to a set of Boolean functions (Fletcher et al. 1993a). Substituting the above inequalities gives rules of the form:

```
IF rise_time < 1.25             IF (pulse_height > 0 975  ∧ pulse_height < 1.025 )
THEN rise_timeflag              THEN pulse_heightflag
ELSE ¬rise_timeflag             ELSE ¬pulse_heightflag
     .                               .
     .                               .
```

These rules serve to illustrate the usefulness of the interpretation system for a medium sized control network. This type of result is most useful when it is necessary to check that what the network has learnt is reasonable and when it is necessary for a system to explain its actions, something classic neural networks cannot normally do.

3 Noise Immunity of Neural Network Derived Rules

A good connectionist based solution should exhibit fewer overall problems than the currently available symbolic systems. A useful quantitative measure of noise immunity is the average percentage of the learnt hypothesis that matches the required hypothesis. A correct function is represented by the truth table shown in table 1a, and a noisy version with one output incorrect is represented in table 1b. As these truth tables match in 87% of the outputs the noisy hypothesis is defined to have an integrity of 87%.

	Table 1a	Table 1b
	00000000	00000000
INPUTS	01010101	01010101
	00110011	00110011
	00001111	00001111
OUTPUTS	01010101	01010111

Table 1a is a truth table representing a Boolean function, whereas table 1b represents a noisy hypothesis, note the change of the seventh bit in the OUTPUTS line.

Figure 1 shows the responses of a neural network after training for conjunction and parity with differing amounts of noise. This demonstrates the immunity to noise using a neural network induction system, although live applications will be more complex than the illustrations above they will typically be trained on noisy data.

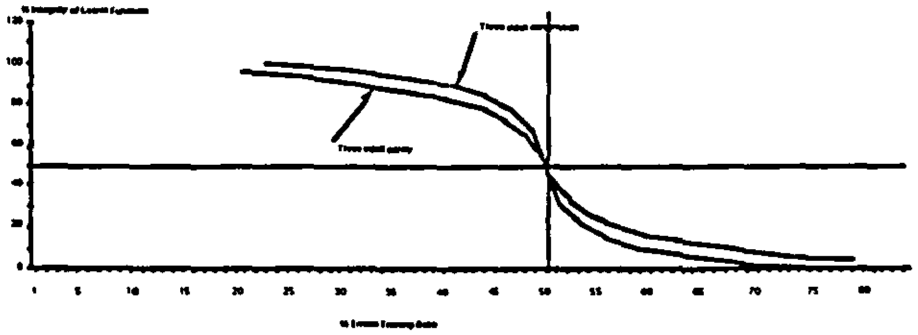


Figure 1 The response of a neural network derived from a training set for conjunction with differing amounts of noise superimposed on the response of the neural network derived from a 3 dimensional parity training set.

4 The Pole Balancing Problem

The pole balancing problem (Michie et al. 1968) is an example of applied adaptive control and has become a standard tutorial problem. The control system must balance a pole on a motorised cart by moving the cart back and forward in a confined space. The implementation of most interest here is the neural network (Zhang et al. 1989) shown in figure 2 which was successful in a balancing the pole under a variety of circumstances.

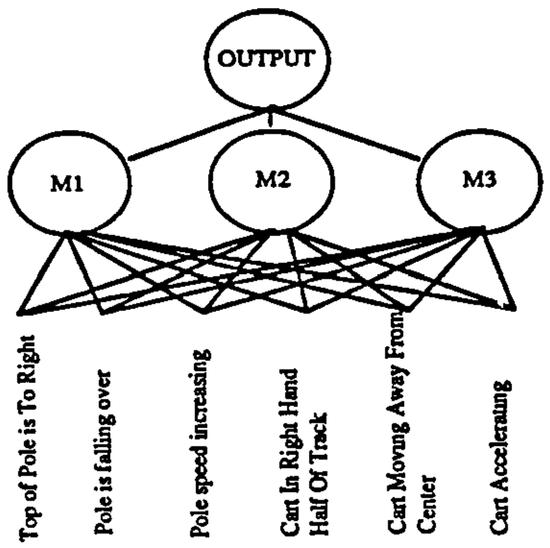


Figure 2. A neural network developed by Zhang and Grant to solve the pole balancing problem.

Fletcher et al. (1993b) describe a method for applying simple image enhancement techniques to Boolean functions in order to expose the underlying emphasis. The analysis of the pole balancing net resulted in the following versions of the output function under different levels of simplification.

IF (Top of pole is to right ∧ ¬Pole is falling over ∧ ¬Pole Speed Increasing ∧ Cart Accelerating) ∨
(Top of pole is to right ∧ ¬Pole is falling over ∧ ¬Pole Speed Increasing ∧ Cart In Right Hand Half Of Track ∧ Cart Moving Away From Centre) ∨
(Top of pole is to right ∧ Pole is falling over ∧ Pole Speed Increasing)
THEN Apply right force
ELSE Apply left force

Set of rules generated with no simplification

IF (Top of pole is to right ∧ ¬Pole is falling over ∧ Pole Speed Increasing) ∨
(¬Top of pole is to right ∧ Pole is falling over ∧ ¬Pole Speed Increasing) ∨
(¬Pole Speed Increasing ∧ ¬Cart In Right Hand Half Of Track ∧ ¬Cart Accelerating)
THEN Apply right force
ELSE Apply left force

This shows us that the function attaches importance to input variables that should have no bearing on the outcome for the cases analysed. This means that the output is not correctly computed.

IF (Top of pole is to right ∧ Pole Speed Increasing) ∨
(Top of pole is to right ∧ Pole is falling over)
THEN Apply right force
ELSE Apply left force

This shows the net has implemented an asymmetric function in response to a symmetric problem. This means that there are some areas that will need further training for completeness.

IF Top of pole is to right
THEN Apply right force
ELSE Apply left force

This shows that the network is *basically* correct.

5 Limits of Boolean Rule Conversion

The nature of neurons means that as the number of inputs grows so does the length of the Boolean description. In the worst case the numbers of conjunctions in a minimal disjunction of conjunctions representation grows at a rate of 2^{n-1} where there are n inputs. The time taken to calculate an answer is exactly proportional to its size. The answer becomes intractable to compute for neurons with more than 20 inputs, and meaningless for a human far earlier. Current work is aimed at implementing systems which will enable the user to interactively train and study the hypotheses of large networks (Fletcher et al. 1993c).

6 Conclusion

Results have been demonstrated that illustrate the extraction of Boolean based rules from a neural network. These rules were also used to check that the hypopthesis of the neural network was consistent with the problem. The rule induction system has also been shown to have a high integrity in the presence of noise, making neural networks an ideal method for rule induction in noisy problem domains.

7 References

Chandraker, R., West, A.A., & Williams, D.J., 1990, Intelligent control of Adhesive Dispensing, IJCIM, special issue in Intelligent Control, 3, No 1, pp. 24-34.

Fletcher, G.P. & Hinde, C.J., 1993a, Interpretation of neural networks as Boolean transfer functions, Dept. Computer Studies research report 779, to be published in Knowledge-Based Systems.

Fletcher, G.P. & Hinde C.J., 1993b, Using neural networks as a tool for constructing rule based systems, Dept. Computer Studies research report 781.

Fletcher, G.P. & Hinde C.J., 1993c, Providing evidence for the hypothesis of large neural networks, Dept. Computer Studies research report 793, submitted in revised form to Neurocomputing.

Hart, A., 1986, Knowledge Acquisition for Expert Systems, Kogan Page.

Messom, C.H., Hinde, C.J., West, A.A. & Williams, D.J., 1992, Designing neural Networks for Manufacturing Process Control Systems, Proc. 1992 International Symposium on Intelligent Control, August 1992, I.E.E.E. Publishers.

Michie, D. & Chambers, R. A., 1968, BOXES: An Experiment in Adaptive Control. Machine Intelligence 2, ed. Dale, E. and Michie, D., Oliver and Boyd, Edinburgh University Press.

Zhang, B. & Grant, E., 1989, A Neural Net Approach to Autonomous Machine Learning of Pole Balancing. Proc. IEEE conference on Intelligent Control. pp 123.

Learning the Activation Function for the neurons in Neural Networks

Proceedings of the International Conference on Artificial Neural Networks, 1994

Maria Marinaro and Pietro Morasso (Eds.)

Springer-Verlag (Pub.)

pp 611 - 614

Learning the Activation Function for the Neurons in Neural Networks

G.P. Fletcher & C.J.Hinde
Department of Computer Studies
Loughborough University
Loughborough

1 Introduction

Ever since the sigmoid replaced the threshold as the main activation function used in artificial neural networks, the properties of the activation function have been largely ignored. Most research aimed at improving the quality of the hypothesis or the speed at which it is obtained has concentrated on the topology or enhancing the learning algorithm (normally back propagation).

Even though the sigmoid function is intended to be an approximation to a real neurons response, artificial systems do not necessarily need to copy. Real neural networks model very complex information by using huge numbers of neurons. By developing a set of more complex activation functions it should be possible to produce better models of complex information resulting in smaller and faster networks and the development of more powerful application areas for artificial neural networks.

Other recent work has presented the idea of parametrisation of the sigmoid function (Horejs et al. 1993). By training these parameters using back propagation it is possible to alter the sigmoid's characteristics; i.e. such things as the maximum and mid values of the function. The idea was developed as a method of trying to remove the input weights as a training parameter. Instead of attempting to produce smaller networks with more powerful nodes the other work was aimed at allowing massive nets of very low complexity to be trained. The results show that although these nets will converge they require many more iterations. This paper shows that, by taking the converse view and improving the power of each neuron, the number of training iterations is significantly reduced.

2 Sine as an activation function for back propagation

Back propagation relies on the use of a differentiable activation function. The sigmoid function was originally chosen because of its similarity to the then popular threshold activation function of the simple perceptron model (Rosenblatt 1962). There is no reason why back propagation needs to use the sigmoid, replacing it with another differentiable function such as sine does not affect the convergent properties but does affect the types of hypotheses that can be represented.

If a two input neuron uses a sigmoid then it is perfectly able to model transfer functions such as "Boolean AND" and "Boolean OR", in fact it is difficult to imagine a better activation function for these two problems.

The "Boolean X-OR" or two input parity is the simplest problem that cannot be modelled in a single neuron using a sigmoid activation function. Figure 1 shows how the sine and weights interact to model the X-OR function in a single neuron.

Case 'A' shows that the threshold with two inputs set to false give 'Sum' a value of $-\pi/2$, and therefore an output value of -1.0 or false.

Case 'B' shows that if one of the inputs is true and the other false then 'Sum" is $\pi/2$, and the output value is 1.0 or true.

Case 'C' shows that with both inputs set to true then sum is $3\pi/2$ and output is -1.0 or false.

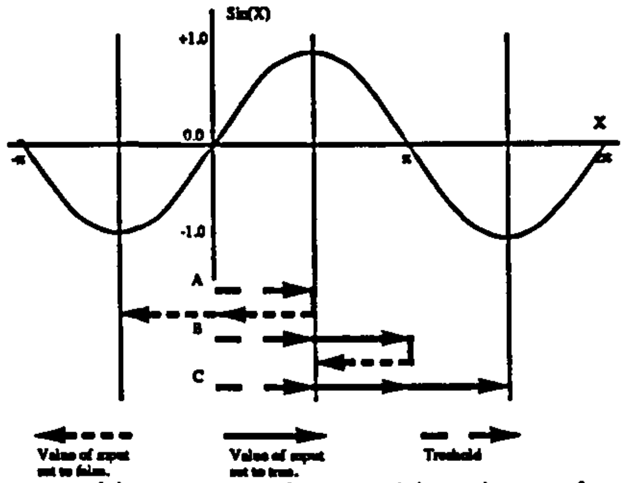


Figure 1 Demonstration of the sine activation function, and the combination of weights and bias to produce the X-OR function.

However, the sine activation function does not effectively model "Boolean OR". Even using such a simple network, it is apparent that by choosing the correct activation function for the problem, modelling becomes much easier.

3 The general activation function

Given a large network it is not feasible for the designer to allocate the correct activation function to every neuron. It therefore becomes necessary to develop a general parametrised activation function with enough flexibility to be able to model many different types of transfer function. If the parameters of the general activation function are trained together with the input weights the neuron will develop the activation function that most easily models the data.

An example activation function is constructed using both the traditional sigmoid and the sin activation functions. Let the activation function be $f(\beta)$ where β is the product of the weight and input vectors. i.e.

$$\beta = \text{bias} + \text{weight}_1 * \text{input}_1 + \text{weight}_2 * \text{input}_2 \text{ ....}$$
$$f(\beta) = a \,.\, \text{sigmoid}(\beta) + (1 - a) \,.\, \sin(\beta)$$

Each element of the activation function has a -1 to +1 range. The output of the neuron has been maintained in the traditional -1 to +1 range by making the sum of the activation function weights equal 1. By back propagating the errors and using them to modify the parameter 'a' in each neuron the individual neurons will develop different characteristics. If a is high, i.e. 0.9, then the activation function becomes very close to the traditional sigmoid. This is very good when attempting to model problems like the "Boolean AND". Dropping the value of a to 0.1 produces an activation function easily capable of representing the "Boolean X-OR" problem.

4 Training with the general activation function with two dimensional problems

All two dimension problems can be represented using just a single neuron with a general activation function. To demonstrate the neurons ability to derive the correct activation function the solutions for "Two input AND" and "Two input X-OR" are presented.

<u>TWO INPUT AND</u>

Input 1 Weight	: +1.7142
Input 2 Weight	: +1.7159
Bias	: -1.7126

Activation Function

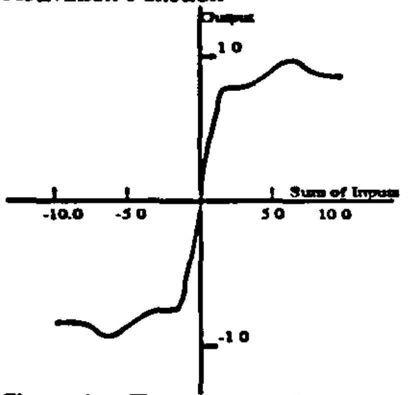


Figure 2a: The activation function in response to the "AND" training set.

<u>TWO INPUT X-OR</u>

Input 1 Weight	: +1.5881
Input 2 Weight	: +1.5776
Bias	: +1.5718

Activation Function

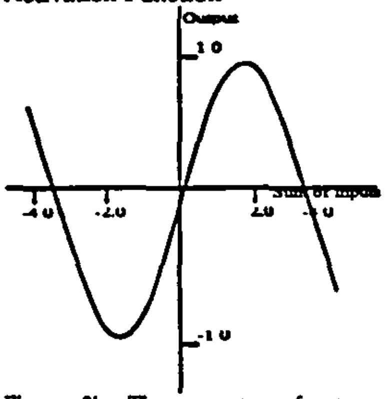


Figure 2b: The activation function in response to the "X-OR" training set.

Both of the neurons have developed activation functions capable of representing the problems being demonstrated. One having produced a threshold type function and the other a cyclic function, ideal for the two different problems.

5 Neurons with general activation function as a method of reducing the number of training iterations required.

Networks constructed of neurons that use the general activation function are more flexible in the representations they can adopt. If it is easier to represent the information within the network, the time required to find an acceptable representation will be reduced.

A four input, three midnode one output network forms the basis for the illustrative experiment, which is shown using standard and general activation functions. The graphs show the error over the training cycle for the two different types of neurons.

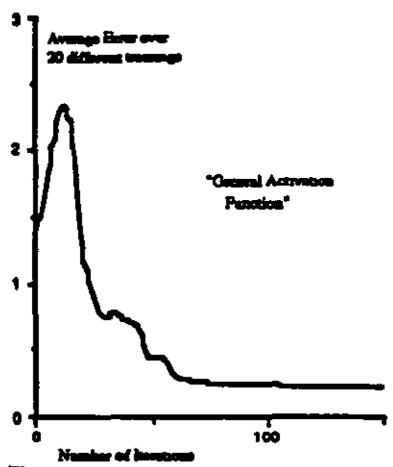


Figure 4a: Average sum squared error when using a general activation function.

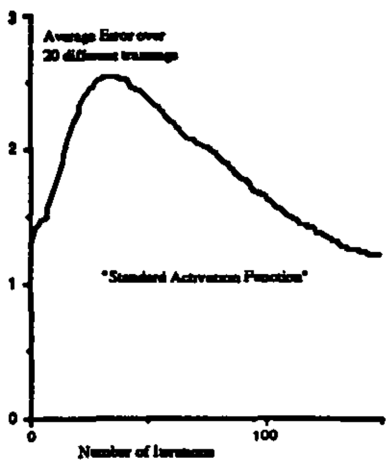


Figure 4b: Average sum squared error when using a standard activation function.

These results show the potential for very large savings in training time. The results are an average of 20 different training cycles. Each one uses a randomly chosen, consistent, training set and start point. The same training sets and starting points were used for both experiments.

6 Conclusion

By increasing the computational power of each neuron it is possible to increase the representational flexibility of neural networks while reducing the number of training iterations required to form a hypothesis.

By reducing the need for very large networks implementation becomes much easier. With the smaller networks it is tractable to produce a clear representation of their derived hypotheses (Fletcher et al. 1993). The developer will then be able to verify that the design has correctly represented the required hypothesis. Greater confidence in neural networks will enable them to be used on a much wider basis.

7 References

Fletcher, G.P., Hinde, C.J., West, A.A. & Williams, D.J., (1994) Neural networks as a paradigm for knowledge elicitation, ibid.

Horejs, J. & Kufudaki, O., (1993) Neural networks with local distributed parameters, Neurocomputing, Vol 5 №4. pp 211 - 219.

Rosenblatt, F., (1962) Principles of neurodynamics, Spartan books.

Using Neural Networks as a tool for constructing rule based systems

Knowledge-Based Systems

Knowledge-Based
SYSTEMS

Prof E. A. Edmonds Editor-in-Chief
LUTCHI Research Centre
University of Technology
Loughborough
Leics LE11 3TU UK

Telephone +44 509 222690
Telex +44 509 610815
E-mail KBS @ lut ac uk

KBS-275

22nd September, 1994

Graham Fletcher
Department of Computer Studies
Loughborough University of Technology
Loughborough
Leicestershire LE11 3TU.

Dear Graham,

Thank you very much for revising your paper entitled, **'Using neural networks as a tool for constructing rule based systems'** for publication in **Knowledge-Based Systems.**

The paper has now been re-examined and as all the referees' comments appear to have been dealt with adequately, I am pleased to inform you that it has been accepted for publication.

Authors receive a copy of the issue of the journal in which their paper is published, plus 50 offprints (more can be ordered if required). Your paper will be published as soon as the schedule will allow and proofs will be sent to you for checking in due course.

In the meantime, please would you sign and return the enclosed copyright transfer form. Assignation of copyright in this way does not jeopardise your traditional rights as an author to reuse or veto third party publication. However, it does enable the publishers to sanction the printing and reprinting of the journal.

If you have not already done so, please also send your original artwork to me to pass on to the publishers (line drawings, prints or negatives).

Yours sincerely

[illegible]

Professor E.A. Edmonds

enc

Producing evidence for the hypothesis of large neural networks

Neurocomputing

Volume 9, 1995

ELSEVIER

Neurocomputing 9 (1995) NCG00362 T

NEUROCOMPUTING

Producing evidence for the hypotheses of large neural networks

G.P. Fletcher, C.J. Hinde *

Dept Computer Studies, University of Technology, Loughborough, Leicestershire LE11 3TU, UK

Received 20 December 1993; accepted 23 September 1994

Abstract

This paper presents results allowing large networks to be analysed using a fast approximate system. This provides an understanding of their hypotheses and actions, permitting networks to be used with more confidence in complex applications. The technique used has been to calculate the best input pattern that given the internal hypothesis gives the desired output. If the calculated pattern correctly displays the features of the input space then this is evidence that the network has learnt the correct hypothesis. The converse is also true: if the shape differs from the target then this should provide some indications for the best training examples to improve the hypothesis.

Keywords: Feedforward networks; Hypothesis; Model; Interpretation: Feedbackward

1. Introduction

One of the major problems preventing the rapid spread of connectionist systems is the lack of a guarantee that the hypothesis represents a correct interpretation of the problem. Hinton [6] in 1990 expresses the view "the problem is to devise effective ways of representing complex structures in connectionist networks without sacrificing the ability to learn the representations. My own view is that connectionists are a very long way from solving this problem".

There are several different methods of describing the output function of a neural network. The purpose for which the description is to be used dictates which method is used. Earlier work has focused on the interpretation of neural networks as rule based systems [4,5], the example being developed here is for the verification of visual pattern recognition networks such as Optical Character Recognition

* Corresponding author. E-mail: c.j.hinde@lut.ac.uk.

SSDI 0925-2312(94)00064-6

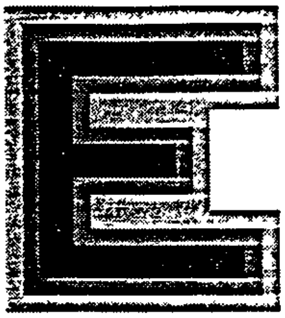


Fig. 1. The target... This is what a feed backward analysis of a network that has correctly learnt to recognise the letter E should look like. The darker the region the closer the link to the hypothesis.

(O.C.R.) [8]. In these types of application the best form of verification is to calculate the input pattern that corresponds most closely to a given output pattern. If this coincides with the developer's ideas then this is evidence for accepting the hypothesis. This would best be described as a form of feed backward network, trying to find the typical input pattern from a given output pattern.

The ideal style of answer would be something along the lines of Fig. 1. The darker regions represent areas that have strong links with the output, lighter areas have a progressively smaller impact. The Fig. 1 hypothesis would be a good result from a network that has been trained to recognise the letter capital E. The Fig. 2 hypothesis could represent a network that correctly classifies all the training examples but could do with some further training. By looking at the result in Fig. 2 it is possible to construct a further training example to correct the minor errors.

2. Boolean rule induction from neural networks

Fletcher and Hinde [4] have produced a system capable of extracting Boolean based rules from a neural network. This system proved very successful when working with small to medium sized industrial control networks.

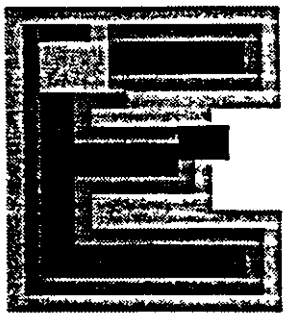


Fig. 2. This is what a feed backward analysis of a network that has nearly correctly learnt to recognise the letter E may look like. Some areas could do with further training examples.