Implementation and Capabilities of Layered Feed-Forward Networks

Gareth D. Richards

Submitted for the degree of

Doctor of Philosophy

Department of Physics
University of Edinburgh

January 1990

To Mandy and my parents

and the memory of Elizabeth Gardner

Declaration

All the work in this thesis is my own except the work in chapter 2 which was done in collaboration with Frank Smieja and has been published in

Smieja F. J., Richards G. D., Complex Systems **2**, 6, 671-704, (1988)

Acknowledgments

My work was funded by the Science and Engineering Research Council and by the Royal Signals and Radar Establishment, Malvern.

I would like to thank Frank Smieja for putting up with me during our work which ended up in chapter 2, David Wallace for helpful suggestions and direction and Mike Brown for accommodating my quota needs. I would particularly like to thank Nick Radcliffe and Frank for their helpful software which was used in the preparation of this thesis. I would also like to thank the people at the RIPR project for making my visits there most enjoyable.

Abstract

This thesis is an investigation into the properties of the layered feed-forward neural network using the back-propagation of error algorithm for training and a description of an implementation of this algorithm on a parallel transputer based computer. The work is in four main parts. In the first part a simple problem is investigated. This demonstrates some of the difficulties that can be encountered during the training of such networks. Methods are developed involving the parameterisation of the training data which allow the problem difficulty to be varied which avoid some of the problems. Considerations of the error surface also lead to suggested improvements to the basic algorithm.

The parallel implementation allows for the simulation of much larger networks than would otherwise have been possible in reasonable times. The code described is used for the simulations in the main body of this work. Methods of dividing the algorithm amongst the transputers are discussed and possibly better methods than that used are suggested.

An investigation of some of the properties of the networks when used as a classifier is discussed. Special attention is paid to the correlation lengths of the images used. These investigations in turn lead to an investigation of the generalising abilities and possible connections are found.

Finally, the properties of the networks when used as associative memories are investigated. The performance of the network is studied when presented with inputs with various overlaps with the patterns on which it was trained. The merits of using information measures to assess performance is discussed which is found in some cases to suggest a possibly different interpretation of the results. Comparison is made between these networks and Hopfield type and several similarities are found.

Contents

Chapter 1

Introduction

Over the last few decades there have been tremendous advances in computer technology. Since the days of ENIAC and the other early machines both their speed and capabilities have increased greatly. One of the limitations that still exists is that computers can only solve problems for which an algorithm or set of rules can be written. This means that computers are good at problems that can be broken down into a set of rules or operations which are evaluated in a sequence of steps in the conventional model of the serial computer. For this reason the silicon computer cannot as yet approach the processing power of the brain in such fields as vision, speech recognition, association, etc. The remarkable power of the brain is achieved despite the fact that the timescale of a typical individual neuronal operation is of the order of one millisecond rather than the nanoseconds of a modern integrated circuit. It is the aim of neural network models to capture the key ingredients responsible for these faculties, amongst which is undoubtedly massive parallelism since at any given time many neurons are altering their states in response to other neurons. That is, the processing power of the brain is distributed over many neurons for any given task. For example in a classification problem there cannot be a purely sequential search but there must somehow be many simultaneous searches.

A neuron consist of a cell body with an output fibre called the axon off which branch the synapses. Electrochemical pulses are fired across this axon and along the synapses which connect the neuron to the input fibres, the dendrites, of other neurons. The human brain consists of some 10^{10} neurons. Each of these neurons

is typically connected to another 10^4 neurons giving roughly 10^{14} connections or synapses. The firing rate of a neuron depends upon its electrochemical potential. When a cell fires the pulses propagate along the axon to the synapses where they induce further pulses in the dendrites of other neurons. These input pulses then either increase or decrease the electrical potential of that neuron depending on whether the synapse is excitatory or inhibitory.

The functioning of the brain on the neuronal level is fairly well understood and some of the main functions have been localised but how the higher level functioning is achieved is still a mystery. The spirit of many neural network models is to try and reproduce the functionality of the brain using a simplified model of the neuron. The problem is similar to that encountered in many areas of Physics, where the macroscopic behaviour of, for example a ferromagnet, can be modeled using a simplified model of the microscopic. The key idea is that relatively simple rules for the behaviour at this microscopic level can then give complex and fairly accurate behaviour at the macroscopic level. Another example is the use of cellular automata like models in the simulation of fluid flow. Here a small number of microscopic collision rules produce realistic macroscopic flow behaviour.

Many models have been developed which differ in structure and detail but share common features which are intended, in the spirit of above, to capture (in an extremely simplified way) a model of the brain. They all contain simple processing units based on the neuron which are described by a real variable representing a firing rate. These units, or nodes, are connected so that the state of a node affects the potentials of all the nodes to which it is connected according to the strength, or weight, of the connection or synapses. The state of the node is then a non-linear function of the potential.

Input to these networks is achieved by setting the states for a subset of the nodes to given input values. These nodes are then referred to as the input nodes and their states as the input vector or pattern. The network then evolves according to some rules which depend on the model until some output activity can be read from another, possibly disjoint, set of nodes. These nodes are called the output nodes and their state is the output vector or pattern.

Neural networks can be thought of as producing mappings from an input space to an output space and this mapping is controlled by the inter-neuron connections or weights. The key to neural networks is to produce this mapping by altering the weight values. How this is done is model dependent and the algorithm used is called the learning algorithm. Learning algorithms can be divided into three main classes. Firstly there are the supervised algorithms where target output vectors are provided. Secondly there are the reinforcement procedures where a measure of the performance of the network is given. Finally there are the unsupervised algorithms where the network captures regularities in the input vectors. The algorithm studied in this thesis is of the first type.

In neural networks the representation of the stored mappings is generally distributed, that is it is represented by a pattern of neuron activity over many neurons and some of the same neurons may fire when a new pattern is presented. Such representations have advantages over local representations. In particular they are generally more efficient since they often require fewer neurons and are more robust to damage: the loss of a few neurons will not completely destroy the functionality of the whole.

1.1 A Brief History

Neural networks as a subject can be said to have had its birth in the 1940's with the publication of two works by McCulloch and Pitts(1943, 1947) who for the first time combined neurological networks with the ideas of finite state machines and linear threshold functions. They showed that any logical function can be duplicated by a network of interconnected linear threshold units: very simple dynamics could generate very complex behaviour. From such ideas developed the world of cybernetics. As a postulate for long term memory Hebb(1949) formulated the basic concept of classical conditioning in terms of cell interactions. Basically he said that if one cell repeatedly takes part in the firing of another then a change takes place in one or both so as to make this interaction more efficient. This became the basis of many neural network learning algorithms. During the 1950's the task became one of trying to make machines that learn tasks rather than

constructing them to perform specific tasks.

In 1962 the perceptron was introduced by Rosenblatt(1962). In the perceptron, the units can be divided into three groups, the input units, the function units and the output or decision units. Each of the function units can apply a different function to part of the input field. The state of the perceptron is then produced by the decision units which apply a linear threshold function of the function units. The perceptron can be thought of as making a decision by adding the evidence obtained from many small experiments (the functions). Rosenblatt introduced many types of perceptron but one of the simplest is where if Φ is the set of functions φ and $\{\alpha\}$ is the corresponding set of weights then the state of the perceptron given some object X is

$$\psi = \text{sgn}\left(\sum_{\varphi\in\Phi} \alpha_\varphi \varphi(X) - \theta\right)$$

where θ is a threshold. The learning process for the perceptron is adjusting the $\{\alpha\}$ and it was shown using the algorithm of Rosenblatt that if such a set exists then it would be found. The learning process is just a Hebb rule; if the sum is too small then increase the offending φ and if it is too big then decrease them.

Much work was carried out with the perceptron and it was found that they could solve many problems but not others. It was also found that they sometimes could not solve problems if they were scaled up. A theoretical analysis of this was published by Minsky and Papert (Minsky and Papert 1969) in which amongst other things they introduced the idea of the order of a perceptron. The order is the smallest value of k for which

$$\forall\varphi \in \Phi \ \ |S(\varphi)| \leq k$$

and ψ is a linear threshold function of the set of functions Φ and $S(\varphi)$ is the set of points upon which φ depends. Minsky and Papert showed that perceptrons could only solve problems of low order. For example, the perceptron cannot in general solve the problem of parity which is to detect whether the number of set bits in the input field is odd or even. In this problem the $\{\alpha\}$ grow worse than exponentially with the size of the problem. Parity is of unbounded order; it is in fact of the order of the input field, and in general for this sort of problem Φ needs to be exponentially big.

Neural network models fall into two main groups: the type in which all the nodes are specified during training and the type in which they are split into several layers. The latter have 'hidden units', the values of which are not specified in the training data. The perceptron can be considered as being of the first type since it has one layer of adjustable weights. Other single layer networks are the Willshaw network (Willshaw 1969) and Hopfield network (Hopfield 1982).

In the Willshaw network the input and the output nodes form disjoint totally interconnected groups with no connections within a group. Neurons and weights are limited to be in two possible states: either zero or one. The weights all start at zero and are set to one if ever both the input node and corresponding output node are set. To recall an output pattern associated with some input pattern the threshold is set to just under the number of active input units.

1.2 The Hopfield Network

The Hopfield type network (Hopfield 1982) usually has total connectivity and the neurons are restricted to be ± 1. In this model the state of a neuron is governed by its local field or potential

$$\phi_i(S) = \sum_j J_{ij} S_j - \theta_i$$

where $\{J\}$ are the interconnection strengths, $\{S\}$ are the node values, the bias θ is generally zero. Generally the constraints $J_{ij} = J_{ji}$ and $J_{ii} = 0$ are imposed. If there is no noise present each spin simply aligns itself with this field so that

$$S_i(t+1) = \mathrm{sgn}(\phi_i(S(t)))$$

but in other models the the new value of the node is given by

$$S_i(t+1) = f\left(\phi_i(S(t))\right)$$

where f is a non-linear function. To make a network do what is wanted, whether it is to do some mapping or to have certain patterns as stable points in the dynamics of the network, the weights or synapses must be set. In the case of the Hopfield network the Hebb rule is used as a one step learning process so that for pattern

S to be stable

$$S_i\phi_i > 0$$

This can be almost achieved by setting the weights

$$J_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu$$

where $\{\xi\}$ is the set of binary patterns to be stored; the nominal patterns.

Hopfield showed that if units are updated sequentially the iterative retrieval process can be viewed as a form of gradient descent of an energy function given by

$$E = -\frac{1}{2}\sum_{ij} S_i J_{ij} S_j + \sum_j S_j \theta_j$$

Each update adopts a state that reduces this energy since if spin S_μ updates $(S_\mu \mapsto S'_\mu)$ according to the above update rule the change in energy is given by

$$\Delta E = -(S'_\mu - S_\mu)\phi_\mu$$

and since $S'_\mu = \mathrm{sgn}(\phi S_\mu)$

$$\begin{aligned} \Delta E &= \begin{cases} \phi_\mu(S_\mu - 1), & \phi_\mu \geq 0; \\ \phi_\mu(S_\mu + 1), & \phi_\mu < 0. \end{cases} \\ &\leq 0 \end{aligned}$$

This type of network has been the subject of much theoretical analysis since the existence of an energy function makes it amenable to methods developed in the statistical mechanics of spin glasses (Amit 1985).

There is a major problem with single layer networks however: they cannot solve certain classes of problems. They can only solve problems which are linearly separable: of the sixteen two bit Boolean functions exclusive or and equivalence are not separable in this way. For example, if the node activities are achieved linearly they cannot discriminate patterns which differ by only one bit.

One of the hopes for multi-layer networks is that they will overcome such problems. In these networks one layer is taken to be the input, and another the output with an arbitrary number of 'hidden' layers in between. The visible units are those set during learning and from which the output of the network is read whereas it is these hidden layers that create the internal representation critical to

problem solving: they find the higher order correlations between the input nodes that cannot be derived by pairwise interaction. For example, if a layered network can be taught to map its input patterns to identical output patterns through a hidden layer with fewer nodes than there are input nodes, the hidden nodes will form a more efficient (since it requires fewer nodes) representation of those patterns.

A type of layered network is the Boltzmann machine, (Hinton 1983), which has a learning algorithm involving probability and annealing. In this model the potential at a node is used to calculate the probability of that node being set. The learning process involves running the network in two ways; the first is with the non-hidden nodes clamped to the desired states and the other is with all the nodes free. Learning is then a method of adjusting the weights so as to minimise a measure of the distance from the probability distribution of the freely running network and the clamped network. A problem with this type of network is that it is very slow to learn.

1.3 Back Propagation

A far more popular type of network (and the type studied in this thesis) is the feed-forward network in which connections only exist in one direction from the input onwards and use a technique called back-propagation of error for learning (see figure 1.1). This technique was first developed by Werbos(1974) and independently discovered by Parker(1982). It however only really became popular after some work by Rumelhart(1985). (A review of the development of back propagation can be found in Werbos(1988).) In this technique, the learning procedure (where the weights are updated) is divided into two parts; the forward and back passes. In the forward pass, the input layer is set to the input pattern and the states of successive layers are calculated until the output layer is set. In the back pass, the output state is compared to its target and from the discrepancy an error can be calculated using a cost or error function. Then, starting at the weights to the output (hence the term back-propagation), derivatives are calculated of the error with respect to each weight. These gradients are calculated using the

chain-rule and summed for all patterns in the training set before the weights are updated. The techniques used is generally gradient descent but other techniques such as conjugate gradient are sometimes used. The process is repeated until some tolerance criterion is met. Generally each node (except for the input) has associated with it a bias. This is a self contribution to the potential and is updated in much the same way as the weights. In 1985 a research group headed by Rumelhart and McClelland published a two volume work (Rumelhart 1985) in which they proposed this type of network which they hoped would overcome many of the problems discussed by Minsky and Papert. For example, in the perceptron the functions do not change: all learning is carried out by varying the weights to the decision units. One of the proposals was to extend learning by adjusting the weights to these units and thereby in effect alter the functions. One problem was the learning algorithm. Some algorithms such as the perceptron convergence algorithm are only possible if the desired states of all the nodes are known before hand. This is not the case if there are hidden nodes. To solve this the chain rule is used as described above to calculate an error derivative for each weight. A problem with the linear threshold functions is that the gradient is zero at all but one point and so a new function must be used. A price has to be paid for all these alterations. It turns out that there is no the guarantee of convergence even if a solution exists since gradient descent is prone to local minima.

1.4 Notation

Before the mathematics of back propagation can be described, the notation used must be defined.

- The number of layers in the network is represented by L. The number of nodes in layer l is given by n_l where $1 \leq l \leq L$ and n_1 is the number of input nodes and n_L is the number of output nodes.
- The symbol α_m is used as a counter over the nodes of layer m.
- For training pattern p, the vector of node states in layer i is given by $\mathbf{v}_i^p$ whose j^{th} element v_{ij}^p is the state of node j in layer i. The input pattern is

$\mathbf{v}_1^p$.

- The matrix of weights connecting layer i to layer k is then given by $\mathbf{w}_k^i$. The weight connecting node j in layer i to node l in layer k is given by w_{kl}^{ij}.

- The vector of potentials of the nodes in layer l for pattern p is given by $\boldsymbol{\phi}_l^p$ and the potential of node k in layer l is ϕ_{lk}^p.

- The vector of biases for layer l is given by $\boldsymbol{\theta}_l$. The j^{th} element of this vector, θ_{lj}, is the bias at node j of layer l. The value of $\boldsymbol{\theta}_1$ (the biases in the input layer) is not needed in the model considered here.

- The vector of target values for the output layer and pattern p is given by $\mathbf{t}^p$. The j^{th} element of this t_j^p is the target value of output node j.

- The vector of error values of the output layer (calculated from the target and state vectors of the output layer) for pattern p is given by $\mathbf{e}^p$ and the error at node j is the j^{th} element of this, e_j^p.

- The total error over all output nodes for a particular pattern p is given by E^p and the total over all output nodes and patterns is given by E.

1.5 Derivations

For the forward pass the states of the input nodes ($\mathbf{v}_1^p$) are set to a pattern vector and the states of the other layers are calculated as follows:
The potentials of the nodes in layer i are given by

$$\boldsymbol{\phi}_i^p = \Phi(\mathbf{w}_i^{i-1}, \mathbf{v}_{i-1}^p, \boldsymbol{\theta}_i) \tag{1.1}$$

where Φ is a differentiable function. The states of the nodes in this layer are then a non-linear function of $\boldsymbol{\phi}_i^p$ and a parameter β

$$\mathbf{v}_i^p = \mathcal{V}(\boldsymbol{\phi}_i^p, \beta) \tag{1.2}$$

The parameter β is used as a 'temperature-like' measure. For example in the activation function used in the following chapters this parameter controls how step-like the activation is or how near the step-function of the perceptron.

For the back pass the value of $\frac{\partial E^p}{\partial w_{kl}^{ij}}$ is needed. That is, the partial derivatives of the error for training pattern p with respect to the weight from node j in layer i to node l in layer k. Before these partial derivatives can be calculated, an error function is needed. That is the function that measures the discrepancy between the output and target output.

$$\mathbf{e}^p = \mathcal{E}(\mathbf{v}_L^p, \mathbf{t}^p) \tag{1.3}$$

As will be seen later, the function $\mathcal{E}$ must be differentiable with respect to both $\mathbf{v}_L^p$ and $\mathbf{t}^p$.

Starting at the weights going to the output layer, these derivatives can be calculated as follows

$$\frac{\partial E^p}{\partial w_{L\alpha_L}^{L-1\alpha_{L-1}}} = \frac{\partial E^p}{\partial \phi_{L\alpha_L}^p} \frac{\partial \phi_{L\alpha_L}^p}{\partial w_{L\alpha_L}^{L-1\alpha_{L-1}}}$$

which can then be expanded to

$$\frac{\partial E^p}{\partial w_{L\alpha_L}^{L-1\alpha_{L-1}}} = \frac{\partial E^p}{\partial v_{L\alpha_L}^p} \frac{\partial v_{L\alpha_L}^p}{\partial \phi_{L\alpha_L}^p} \frac{\partial \phi_{L\alpha_L}^p}{\partial w_{L\alpha_L}^{L-1\alpha_{L-1}}}$$

Then with the definition

$$\delta_{m\alpha_m}^p = \frac{\partial E^p}{\partial \phi_{m\alpha_m}^p} \tag{1.4}$$

this can be re-written as

$$\frac{\partial E^p}{\partial w_{L\alpha_L}^{L-1\alpha_{L-1}}} = \delta_{L\alpha_L}^p \frac{\partial \phi_{L\alpha_L}^p}{\partial w_{L\alpha_L}^{L-1\alpha_{L-1}}} \tag{1.5}$$

where

$$\delta_{L\alpha_L}^p = \frac{\partial v_{L\alpha_L}^p}{\partial \phi_{L\alpha_L}^p} \frac{\partial E^p}{\partial v_{L\alpha_L}^p} \tag{1.6}$$

Then the other derivatives can be calculated using the following

$$\begin{aligned}
\frac{\partial E^p}{\partial w_{m\alpha_m}^{m-1\alpha_{m-1}}} &= \frac{\partial E^p}{\partial \phi_{m\alpha_m}^p} \frac{\partial \phi_{m\alpha_m}^p}{\partial w_{m\alpha_m}^{m-1\alpha_{m-1}}} \\
&= \delta_{m\alpha_m}^p \frac{\partial \phi_{m\alpha_m}^p}{\partial w_{m\alpha_m}^{m-1\alpha_{m-1}}} \\
\delta_{m\alpha_m}^p &= \frac{\partial v_{m\alpha_m}^p}{\partial \phi_{m\alpha_m}^p} \frac{\partial E^p}{\partial v_{m\alpha_m}^p} \\
&= \frac{\partial v_{m\alpha_m}^p}{\partial \phi_{m\alpha_m}^p} \sum_{\alpha_{m+1}} \frac{\partial E^p}{\partial \phi_{m+1\alpha_{m+1}}^p} \frac{\partial \phi_{m+1\alpha_{m+1}}^p}{\partial v_{m\alpha_m}^p} \\
&= \frac{\partial v_{m\alpha_m}^p}{\partial \phi_{m\alpha_m}^p} \sum_{\alpha_{m+1}} \delta_{m+1\alpha_{m+1}}^p \frac{\partial \phi_{m+1\alpha_{m+1}}^p}{\partial v_{m\alpha_m}^p}
\end{aligned} \tag{1.7}$$

Using this the gradients from layer m can therefore be calculated from the values of $\delta^p_{m+1\alpha_{m+1}}$. So starting at the output layer, all the gradients can be calculated.

Similarly, for the biases $\theta_{m\alpha_m}$, the gradients are calculated.

$$\begin{aligned}\frac{\partial E^p}{\partial \theta_{m\alpha_m}} &= \frac{\partial E^p}{\partial \phi^p_{m\alpha_m}} \frac{\partial \phi^p_{m\alpha_m}}{\partial \theta_{m\alpha_m}} \\ &= \delta^p_{m\alpha_m} \frac{\partial \phi^p_{m\alpha_m}}{\partial \theta_{m\alpha_m}} \end{aligned} \tag{1.8}$$

Once the gradients have been calculated, the weights can be updated. In its simplest form, gradient descent may be used. That is

$$\Delta w^{m-1\alpha_{m-1}}_{m\alpha_m} \propto \frac{\partial E^p}{\partial w^{m-1\alpha_{m-1}}_{m\alpha_m}}$$

More generally, however, the updating would be more complex.

$$\Delta w^{m-1\alpha_{m-1}}_{m\alpha_m} = \mathcal{W}(\frac{\partial E^p}{\partial w^{m-1\alpha_{m-1}}_{m\alpha_m}}, \ldots) \tag{1.9}$$

where the ...'s are meant to signify some, as yet, unspecified parameters.

In the above, the gradients have been calculated for one training pattern. There is however usually more than one such pattern. The above formulae can easily be generalised. Generally one of two systems is used. Firstly, the weights are updated after each pattern presentation, in which case the above description is correct. Secondly, the gradients are summed for each pattern and the weights are updated using these summed values. In the second case the gradients used would be

$$\frac{\partial E}{\partial w^{m-1\alpha_{m-1}}_{m\alpha_m}} = \sum_p \frac{\partial E^p}{\partial w^{m-1\alpha_{m-1}}_{m\alpha_m}} \tag{1.10}$$

Unless otherwise stated the actual forms used for $\mathcal{E}$, Φ, $\mathcal{W}$ and $\mathcal{V}$ are as follows

$$e^p_j = \frac{1}{2}(v^p_{Lj} - t^p_j)^2 \tag{1.11}$$

$$v^p_{j\alpha_j} = \frac{1}{1 + \exp(-\beta \phi^p_{j\alpha_j})} \tag{1.12}$$

$$\phi^p_{j\alpha_j} = \sum_{\alpha_i} w^{i\alpha_i}_{j\alpha_j} v^p_{i\alpha_i} + \theta_{j\alpha_j} \tag{1.13}$$

$$\Delta w^{ij}_{kl}(n) = -\eta \frac{\partial E}{\partial w^{ij}_{kl}} + \alpha \Delta w^{ij}_{kl}(n-1) \tag{1.14}$$

$$\tag{1.15}$$

where n counts the weight updates.

These forms are those suggested by Rumelhart and have many interesting properties as well as drawbacks. Some of the problems with these forms are discussed in chapter 2. The error function (equation 1.11) is the simplest that preserves the symmetry between positive and negative differences, though other functions are used. The activation function (equation 1.12) is discussed by Rumelhart and has the advantage that it has a particularly simple first derivative. The possible merits of some alternatives are discussed by Dodd(1989) and Bedworth(1988). The potential function (equation 1.13) is the simplest form of potential function and also has very simple derivatives but is not suitable for certain problems. An alternative is used in chapter 2. The weight changing function (equation 1.14) uses two parameters η the 'step size' measure and α the 'momentum' or 'acceleration' which adds a proportion of the previous weight change to the present weight change. The simplicity of this equation can (and often does) cause problems. There have been many ways suggested to improve it (see for example chapter 2). One of the main problems is that the step size is constant for each weight regardless of the form of the error surface in that dimension. The introduction of α is an attempt to remedy this. If the gradient is small then this parameter will tend to accelerate the system in that direction, however if the weight in that direction is in the form of a valley so that the weight change causes the gradient to change sign (that is cross the valley) then the addition of a fraction of the previous change will help pull the system down. This, though not perfect, does help. A situation where this can cause problems is discussed in chapter 2. One of the problems is that α and η are not independent so the choice of one will affect the optimal value of the other (Tollenaere 1989). Other attempts to remedy this have involved using a conjugate gradient algorithm, radial basis functions (Broomhead 1988) and second order terms (Parker 1987).

One of the main problems encountered in gradient descent is that of local minima. The aim in learning is to find the best set of weights which minimise the error function. There will however be many sets of weights that will give low values of error and it will be very simple for the system to find these and not the desired optimum. This problem is discussed in chapter 4.

From equations 1.11 - 1.14 with 1.5 the form of the gradient for output layers

can be derived:

$$\delta^p_{L\alpha_L} = v^p_{L\alpha_L}(1 - v^p_{L\alpha_L})(v^p_{L\alpha_L} - t^p_{\alpha_L}) \tag{1.16}$$

and for other layers

$$\delta^p_{m\alpha_m} = v^p_{m\alpha_m}(1 - v^p_{m\alpha_m}) \sum_{\alpha_{m+1}} \delta^p_{m+1\alpha_{m+1}} w^{m\alpha_m}_{m+1\alpha_{m+1}} \tag{1.17}$$

In general

$$\frac{\partial E^p}{\partial w^{m-1\alpha_{m-1}}_{m\alpha_m}} = v^p_{m-1\alpha_{m-1}} \delta^p_{m\alpha_m} \tag{1.18}$$

From equation 1.8 the gradients for the biases are found to be

$$\frac{\partial E^p}{\partial \theta_{m\alpha_m}} = \delta^p_{m\alpha_m} \tag{1.19}$$

since

$$\frac{\partial \phi^p_{m\alpha_m}}{\partial \theta_{m\alpha_m}} = 1$$

1.6 Chapters

Many problems are encountered during the training of neural networks. Chapter 2 describes an investigation into a simple problem which demonstrates some of these problems. During this work two methods were developed which improved the basic algorithm outlined above. In particular a method was developed whereby the training data is parameterised so that this parameter is used to vary the problem difficulty. In this chapter the role of the form of the error surface is discussed. Considerations of this lead to suggested improvements to the algorithm.

Simulating neural networks is computationally very intensive. For each training pattern there are forward and back passes, each of which involves many floating point calculations. One solution to such problems is the use of parallel computers. Chapter 3 describes an implementation of the back propagation algorithm on a transputer machine and discusses other ways of distributing the algorithm. The work described in chapters 4 and 5 was carried out using this simulator.

Much of the interest in neural networks is in their use as classifiers. Chapter 4 deals with an investigation into pattern classification and the generalising capabilities of these networks. The network was trained on small samples of four digitised images of textures and then tested on other samples. This was to test whether the

network had generalised rather than just learned the images presented. The role of correlation lengths in the images and how they affect the ability of the network to generalise is discussed.

Chapter 5 deals with the use of layered networks as associative memories. Here the storage capacity of the network is studied as a function of the overlap between the input and target output and the information storage capacity of the network is investigated. Much work has been put into investigating Hopfield type networks as such memories and this chapter is by way of a comparison.

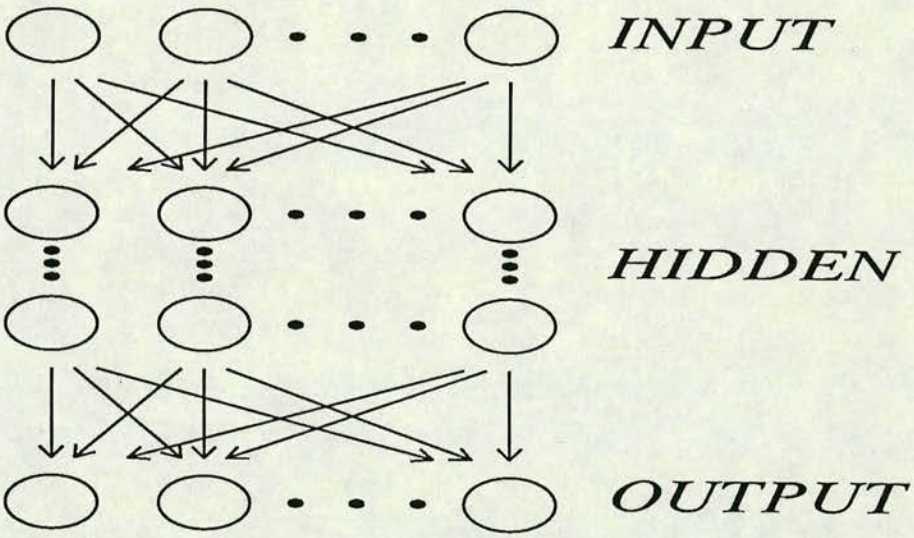

Figure 1.1: An example of a layered feed-forward network.

Chapter 2

Pattern Annealing

The main attraction of neural networks is that they can 'learn' from example, that is the weights are adjusted so that an error is reduced and a desired mapping for some training set is achieved. If this is successful it is often hoped that the network will produce some desired output when presented with some previously unseen input: it will have produced the correct weight matrices for the whole problem from the subset of the data. The choice of the training data and its representation can be crucial to the speed of learning or it can even determine whether the problem can be learned or not. It is for example more difficult to discriminate training patterns if they are very close in the training set but the desired output patterns are very different. This is due to the fact that the functions used in the network are continuous and if similar inputs are mapped to similar outputs [1] then the easier the response of the network. It is also necessary to ensure that the training data is representative of the problem as a whole if the network is to generalise and not just to learn the training set. An example of how controlling the training data can radically improve learning is given below.

The basic idea behind this is to find some method of parameterising the training data so that varying this parameter, which will be called r, takes the problem from an 'easy' problem to that desired. The value of an input node is then a function of this parameter, the pattern number p and the node index i. That is $v_{1i}^{p} = P(p, r, i)$. In the problem studied here the network was required to produce

[1] This is the function the network performs on the input to produce the output.

at the output layer the rounded values of the input nodes. That is

$$v_{Li}^p = \begin{cases} 1, & \text{if } v_{1i}^p \geq 0.5; \\ 0, & \text{otherwise.} \end{cases}$$

where v_{Li}^p is an output node and v_{1i}^p is the corresponding input node. The input patterns are such that

$$\forall i, p, \; v_{1i}^p \in [0, 1]$$

This task has the problem described above where similar inputs must be mapped to dissimilar outputs. Here in fact the inputs can become arbitrarily close as $v_{1i}^p = 0.5 + \delta_i$ and $v_{1i}^p = 0.5 - \delta_i$ must be mapped to 1 and 0 and this becomes increasingly difficult as $\delta_i \to 0$ since the form of the output must approach a step function.

This problem has the feature that each element of the input pattern is totally independent of the other elements (a first order problem in the terminology of Minsky and Papert). Similarly, output nodes only depend on their corresponding input nodes. It would therefore be expected that the solutions would be non-intersecting paths from input to corresponding output nodes. For such paths to exist the hidden layer must have at least the same number of nodes as the input layer. Networks with fewer nodes in the hidden layer would be forced to try and extract correlations between the input nodes, which do not exist; solutions will therefore not exist for such networks. In the case where solutions do exist, the weights on the paths must grow increasingly large as $\delta_i \to 0$ and the biases must tend to zero. This can be shown as follows. Suppose the input to input node v_{1i}^p is $0.5 + \delta_i$ then the state of node v_{Li}^p should be greater than 0.5 and should approach one as δ_i increases for positive δ_i. Similarly if δ_i is negative then v_{Li}^p should be less than 0.5. In this work, a variation of the potential function (equation 1.13) has been used. This change is discussed later. For a three layer network then

$$\begin{aligned} \phi_{2k}^p &= \sum_j w_{2k}^{1j}(2v_{1j}^p - 1) + \theta_{2k} \\ &= 2\delta_i w_{2k}^{1i} + \sum_{j \neq i} w_{2k}^{1j}(2v_{1j}^p - 1) + \theta_{2k} \end{aligned} \tag{2.1}$$

For the output layer

$$\phi_{3i}^p = \sum_l w_{3i}^{2l}(2v_{2l}^p - 1) + \theta_{3i}$$

$$= \sum_l w_{3i}^{2l} \tanh \frac{1}{2}\phi_{2l}^p + \theta_{3i} \tag{2.2}$$

For rounding, ϕ_{3i}^p must be of the same sign as δ_i regardless of the values of the other input nodes. Therefore, if the weight from a node in the intermediate layer to node i in the output layer is non-zero then the terms in equation 2.1 not dependent on that weight must sum to zero. Similarly if this sum is not zero, the weight must be zero.

$$\begin{aligned} \forall l, i \quad & w_{3i}^{2l} \neq 0 \Rightarrow \sum_{j \neq i} w_{2l}^{1j}(2v_{1j}^p - 1) + \theta_{2l} = 0 \\ & \sum_{j \neq i} w_{2l}^{1j}(2v_{1j}^p - 1) + \theta_{2l} \neq 0 \Rightarrow w_{3i}^{2l} = 0 \\ & \theta_{3i} = 0 \end{aligned}$$

Also since the terms of $\boldsymbol{\theta}_2$ are pattern independent they must tend to zero. For the sums to be zero for all input node values the weights must be zero. This therefore gives

$$\forall l, i \;\; w_{3i}^{2l} \neq 0 \Rightarrow \forall j \neq i \; w_{2l}^{1j} = 0$$

Also as $\delta_i \rightarrow 0$ the ϕ_{3i}^p must still be non-zero and since it must be of the same sign as δ_i both w_{2j}^{1i} and w_{3i}^{2j} must become increasingly positive or negative. It can also be seen that the above conditions can only be satisfied if there are at least the same number of hidden nodes as input.

A feature of these layered networks is that any permutation of the hidden nodes produces a functionally equivalent network. The number of possible solutions will therefore increase with the number of hidden nodes. If there are n hidden nodes in the network, the number of possible solutions is the number of ways of arranging the nodes (i.e. $n!$). It would be expected that the learning time would decrease as the number of hidden nodes increases. It should however be noted that the total time taken per cycle will increase so there may not actually be a net decrease in the total time taken to learn the problem. The solution chosen depends on the initial random weights and on the learning parameters chosen. The dependence on the random weights will be discussed later in this chapter.

The method of parameterising the patterns is to omit a region around 0.5. The interval $[0, 1]$ is now divided into three regions, the rejected region $(0.5 - r, 0.5 + r)$

and the regions $[0.5+r, 1.0]$ and $[0.0, 0.5-r]$. The desired mapping now becomes

$$v_{Li}^{p} = \begin{cases} 1, & \text{if } v_{1i}^{p} \in [0.5+r, 1.0]; \\ 0, & \text{if } v_{1i}^{p} \in [0.0, 0.5-r]. \end{cases}$$

When $r = 0$ the problem is the original rounding problem. However as $r \rightarrow 0.5$ the problem becomes a direct mapping from input to output. The direct mapping problem should be a lot easier to learn since the input node values being mapped to different output node values are as different as possible.

2.1 The Training Set

The first problem to be solved is choosing the training data. For $r < 0.5$ there are an infinite number of possible patterns and so the entire training set cannot be presented. One approach is to choose a number at random. But how many? There must be enough to ensure that no appreciable correlations exist so that the data is representative of the whole problem. Another and more satisfactory approach is to construct the training data so that all types of pattern are represented equally. The method chosen was to construct the training data so that it contained all permutations of 1, 0, $0.5+r$ and $0.5-r$ with only one of the second two being present in any pattern. This is sufficient since for each node the data contains $0.5+r$ and $0.5-r$ with all combinations of 1's and 0's on either side. If a number just below 0.5 can be rounded with 1's or 0's on either side then it could be rounded with numbers just above or below 0.5 on either side. This is the minimum training set that ensures that all the weights not on the paths connecting an input node to its corresponding output node will be made to tend to zero. To achieve this it is necessary to ensure that there are no possible weight combinations that would give zero sums in equations 2.2. Giving all permutations of 1's and 0's does this. A sample of such data is given in table 2.1.

Figure 2.1 shows the distribution of learning times for $r = 0.3$ with a 4-4-4 network.[2] This distribution shows a clear starting point at 200 cycles and a very clear peak at 230-240. There are also several runs that fall outside the main peak.

[2]These runs were done with η and $\alpha = 0.3$. Repeated runs were made with different random starts and learning was halted when all output nodes were within 0.1 of their target values.

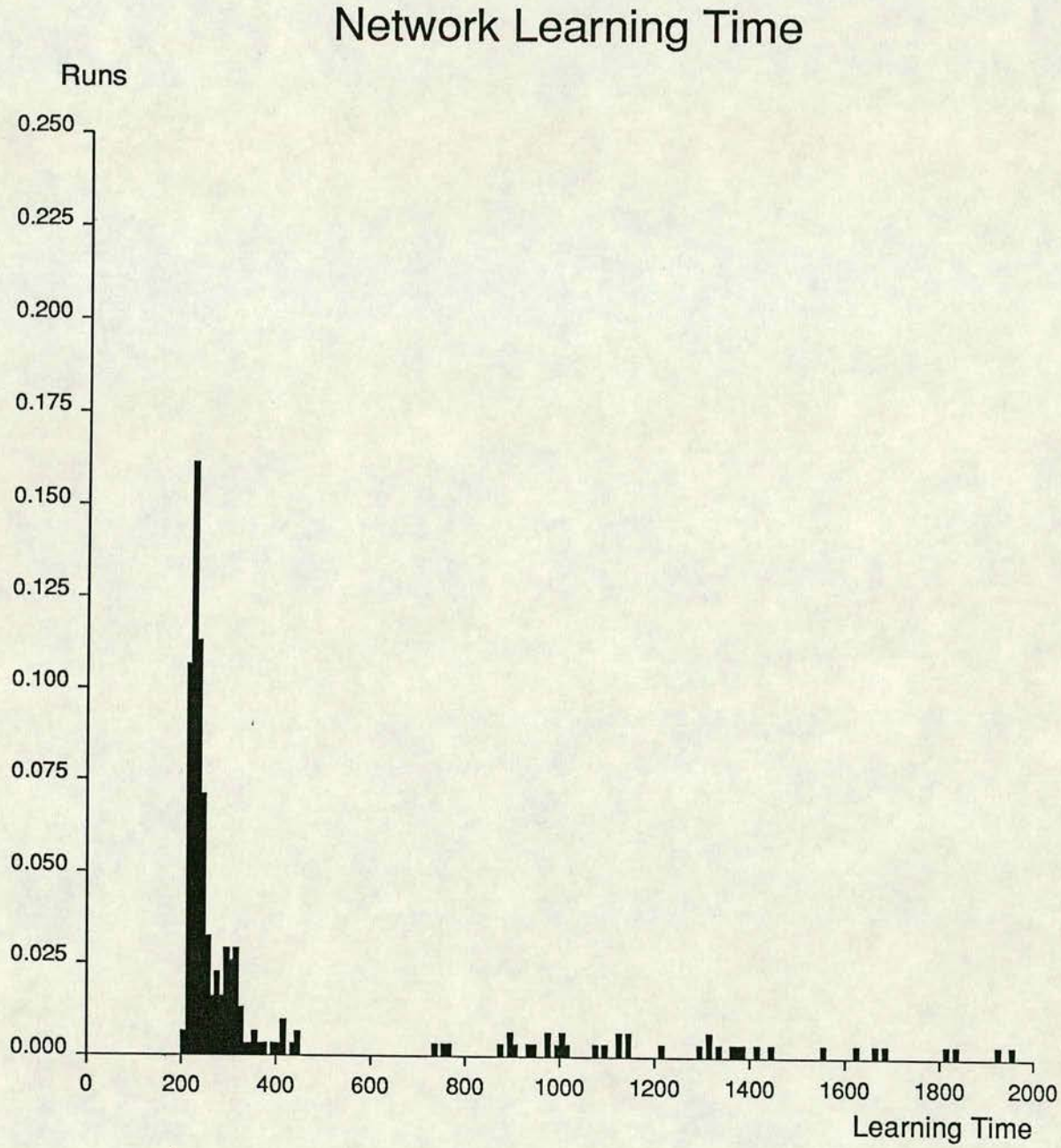

Figure 2.1: The distribution of learning times for a three layer network with four nodes in each layer and $r = 0.3$. The vertical scale shows the fraction of the total (310) runs made. Each of the runs was with a different random start.

unit 1	unit 2	unit 3
R_+	1.0	1.0
R_-	1.0	1.0
R_+	0.0	1.0
R_-	0.0	1.0
R_+	1.0	0.0
R_-	1.0	0.0
R_+	0.0	0.0
R_-	0.0	0.0

Table 2.1: Part of the training data for a three input node network. The rest of the data is obtained by permuting the columns. $R_+ = 0.5 + r$ and $R_- = 0.5 - r$.

This indicates that the minimum time in which the system can learn is around 200-210 cycles and that the mode is between 230 and 240. It was found that approximately 19% of the runs carried out failed to converge. If this is compared with the distribution for $r = 0.0075$ in figure 2.2 it can be seen that far fewer of the runs converged (in fact only 43% of them did) and that a much larger proportion of them are not in the main peak which is much broader than that in figure 2.1. This indicates that the problem is much harder, that is the route to the minimum is more tortuous, so that it is much less likely that the system will find the shorter route to the solution.

The size of the training set here grows exponentially. In fact for i input nodes there are $i2^i$ patterns. A similar scaling problem is found in the parity problem (Rumelhart 1986). This scaling is due to the independence of the nodes. The network cannot deduce the state of one node from the others, so all cases must be presented.

It was mentioned above that the output function must approach a step function, this is illustrated quite clearly in figure 2.3. Here is shown the function near 0.5 for several values of r. The network is fully trained on data constructed as above and then tested on input where all the nodes but one are set to one. The value of the other node is varied from 0.48 to 0.52 and the value of the corresponding output node is read off. It can be seen that the function becomes more

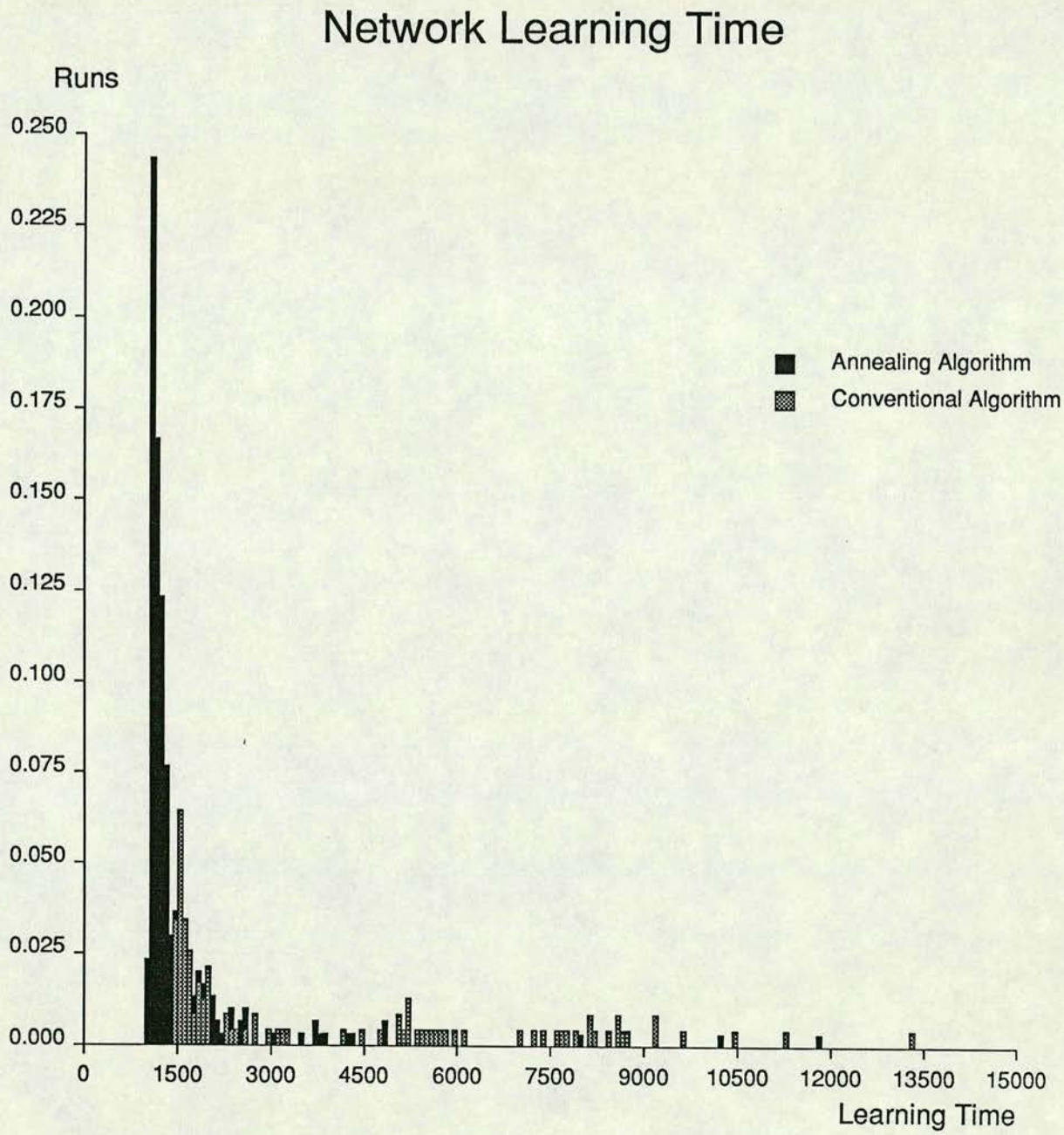

Figure 2.2: The distribution of learning times for both the basic algorithm and annealing for $r = 0.0075$. The vertical scale shows the fraction of the total (233 in the normal case and 300 in the annealed case) runs made that fell within the range.

step-like as r decreases.

2.2 The Basic Algorithm

As was stated earlier, a slight variation of the potential function given in equation 1.13 has been used in this work. This problem illustrates well how a change in this function can greatly improve learning times of the network. The actual function used was

$$\begin{aligned}\phi^p_{j\alpha_j} &= \sum_{\alpha_i} w^{i\alpha_i}_{j\alpha_j}(2v^p_{i\alpha_i} - 1) + \theta_{j\alpha_j} \\ &= 2\sum_{\alpha_i} w^{i\alpha_i}_{j\alpha_j} v^p_{i\alpha_i} - (\theta_{j\alpha_j} - \sum_{\alpha_i} w^{i\alpha_i}_{j\alpha_j})\end{aligned}$$

This change can therefore be thought of as a change in the bias. It is however more sensitive to the state of a node being less than or greater than 0.5 in that it changes sign. The change of this function alters equations 1.17 and 1.18. These are now

$$\delta^p_{m\alpha_m} = v^p_{m\alpha_m}(1 - v^p_{m\alpha_m}) \sum_{\alpha_{m+1}} \delta^p_{m+1\alpha_{m+1}} 2w^{m\alpha_m}_{m+1\alpha_{m+1}}$$

and

$$\frac{\partial E^p}{\partial w^{m-1\alpha_{m-1}}_{m\alpha_m}} = (2v^p_{m-1\alpha_{m-1}} - 1)\delta^p_{m\alpha_m}$$

The effect of this can be seen in figure 2.4. It can clearly be seen that the learning time is significantly reduced. With the conventional potential function the network could not be trained to values of r less than 0.01.[3]

The performance of the algorithm in finding solution can be seen from a plot of the total output error (that is E) against training cycles. Figure 2.5 shows the learning curves for the early stages for various networks and values of r. When the error remains essentially steady it is assumed that the algorithm is unable to converge to the global minimum. It can be seen that for $r = 0.5$ the system finds this minimum quickly. The descent is marked in all the systems by being relatively steep in the first 10 to 100 cycles, followed by a small gradient until the end. For $r = 0.01$, there is again this rapid decrease in error at the start followed

[3]These runs were made with $\eta = 0.1$ and $\alpha = 0.6$

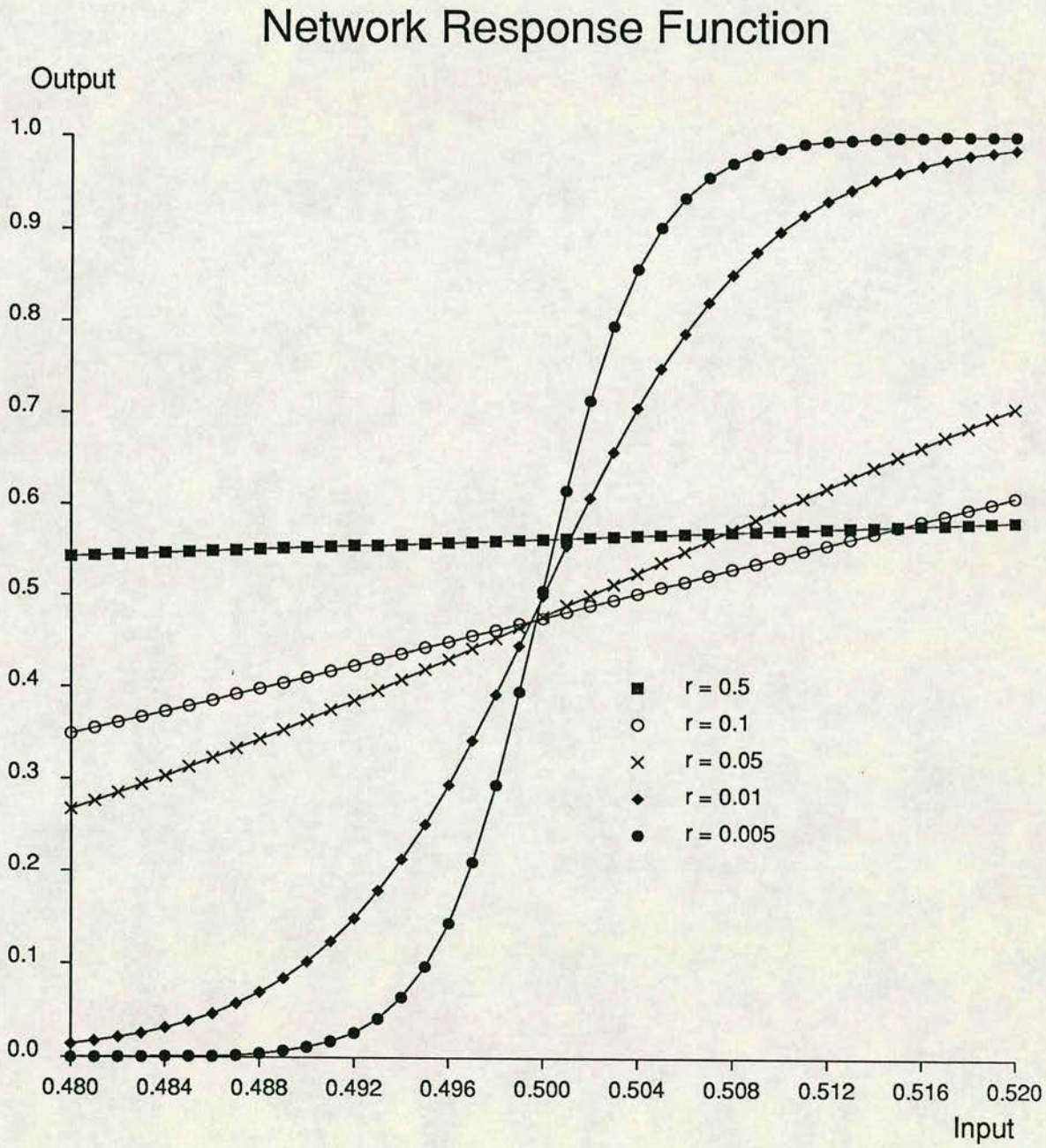

Figure 2.3: The response function for a network trained at various values of r.

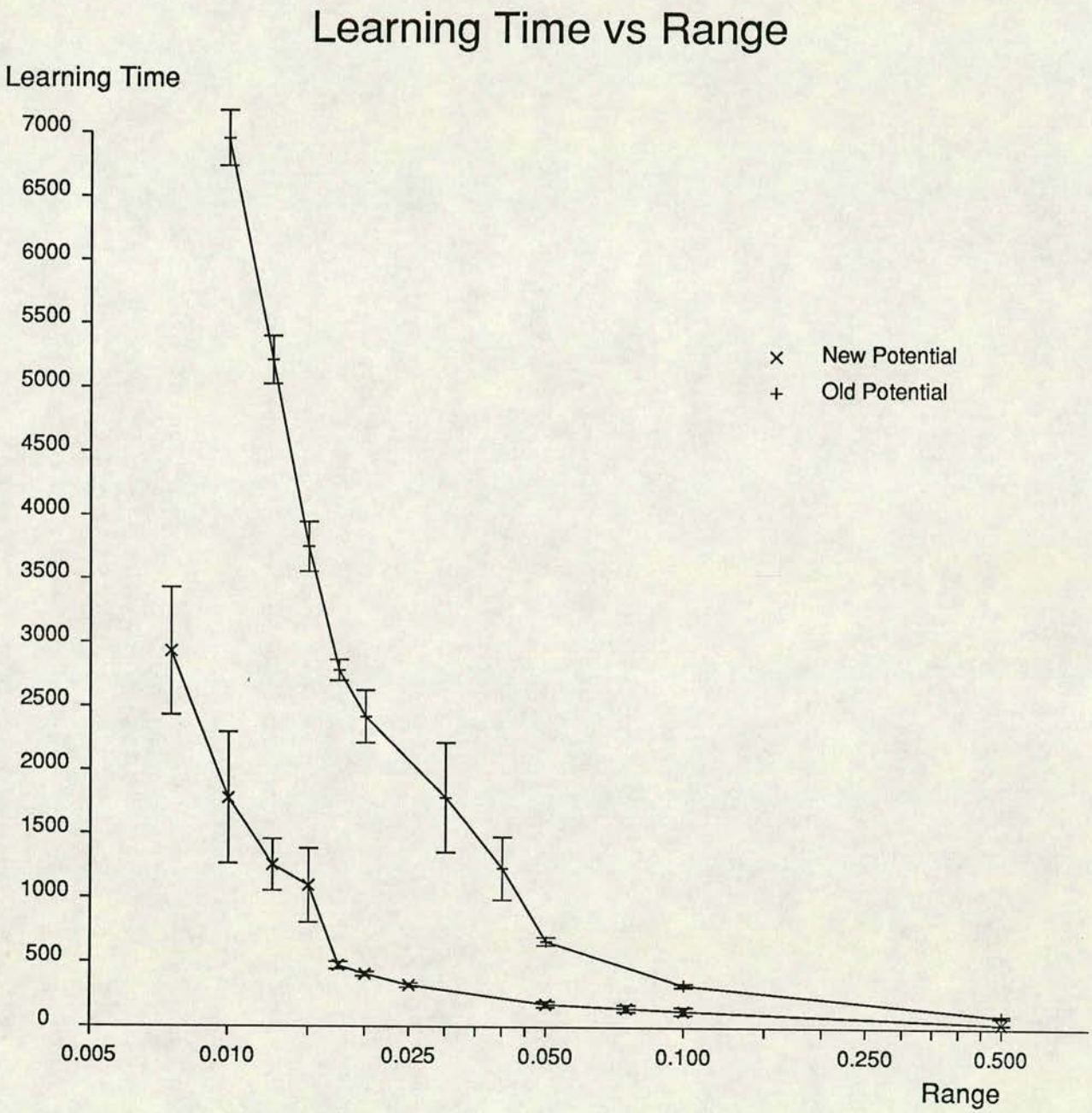

Figure 2.4: The length of time the basic algorithm took to learn to tolerance 0.1 for various values of the parameter r.

by a region of low gradient but there is a steep drop at the end. This drop is more noticeable with the larger networks. For $r = 0.0001$ the steep slope of the start is followed by a level region; none of the systems manage to find a solution.

2.3 The Error Surface

The reason for this increased learning time can be seen from the error surface. The error surface is a way of visualising gradient descent. The dimensions of this space consist of each weight and bias, each point has a value associated with it equal to the total error the network would have with those weight values. It is often helpful to think of this value as a height and gradient descent can then be thought of as moving along valleys and surfaces. A point on this surface therefore defines a network for the particular training data. The error surface plots shown in figures 2.6, 2.7 and 2.8 show the variation of the error in the direction of particular weights. They can be thought of as slices through the error space. These plots were made by halting the learning and calculating the error with adjusted weight values.

As was stated earlier, to solve this problem the network has to find non intersecting paths from input nodes to the corresponding output nodes. A pictorial representation of this can be seen from figure 2.9 which clearly shows these paths. In this example the network has been trained down to a range of 10^{-4} and the magnitude of the weights are represented by the degree of shading. This structure produces three classes of weights, the biases, the dominant weights (i.e. those of the paths) and the small weights (i.e. those off these paths). Figure 2.8 shows the error surface in the direction of a bias near the global minimum. Figures 2.6 and 2.7 show these for the large and small weights.

It can be seen from figure 2.8 that the error surface in the direction of the biases becomes increasingly crevasse-like with decreasing r. This is the cause of the problem. If the network is trained with small r it becomes increasingly difficult for the network to find this shrinking minimum. If figure 2.6 is compared with figure 2.7 for the small weights it can be seen clearly that for the small weights the width of the minimum changes little with r. The surface for the large weights

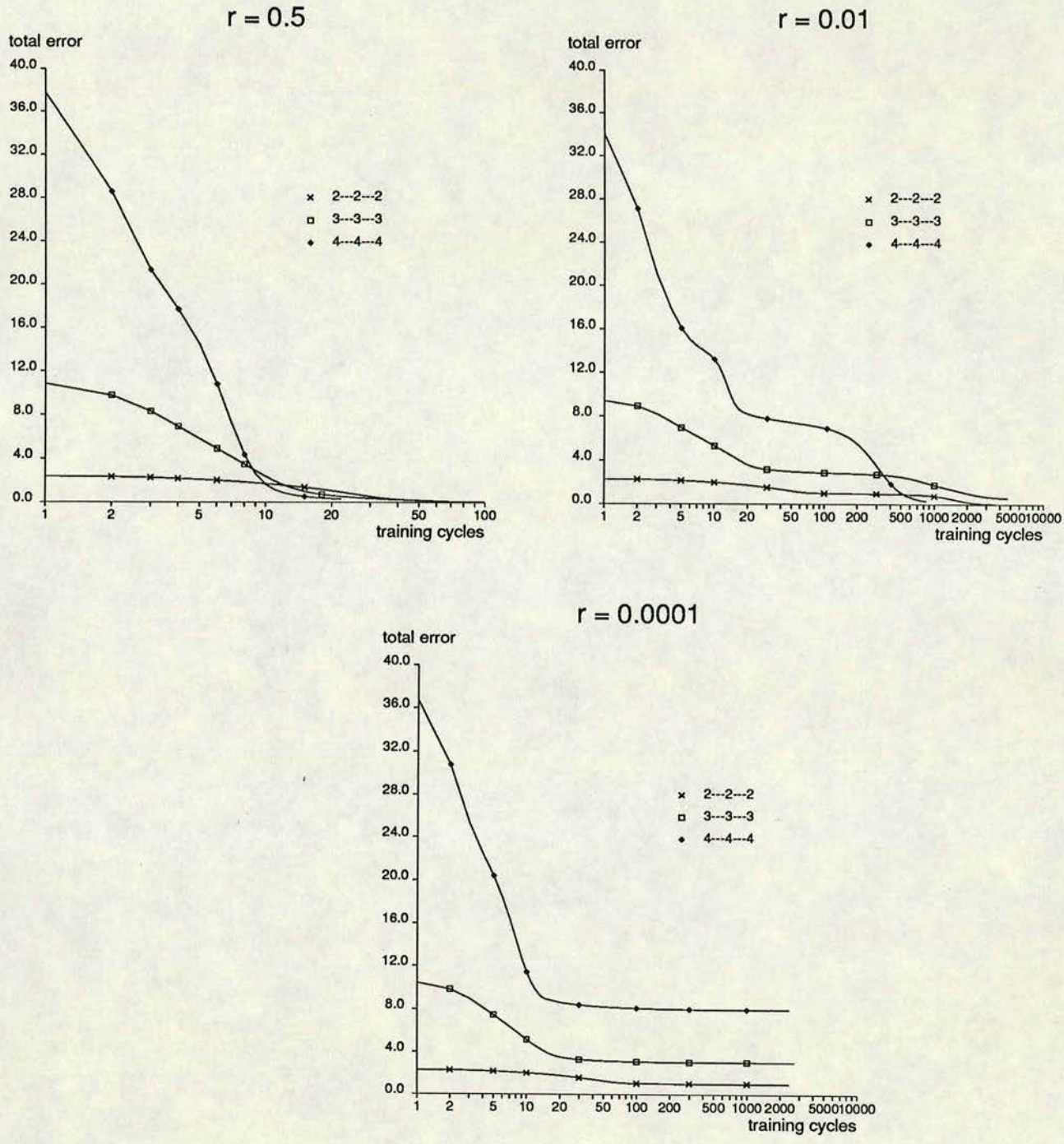

Figure 2.5: The progress of the error with training time for various network sizes and values of r.

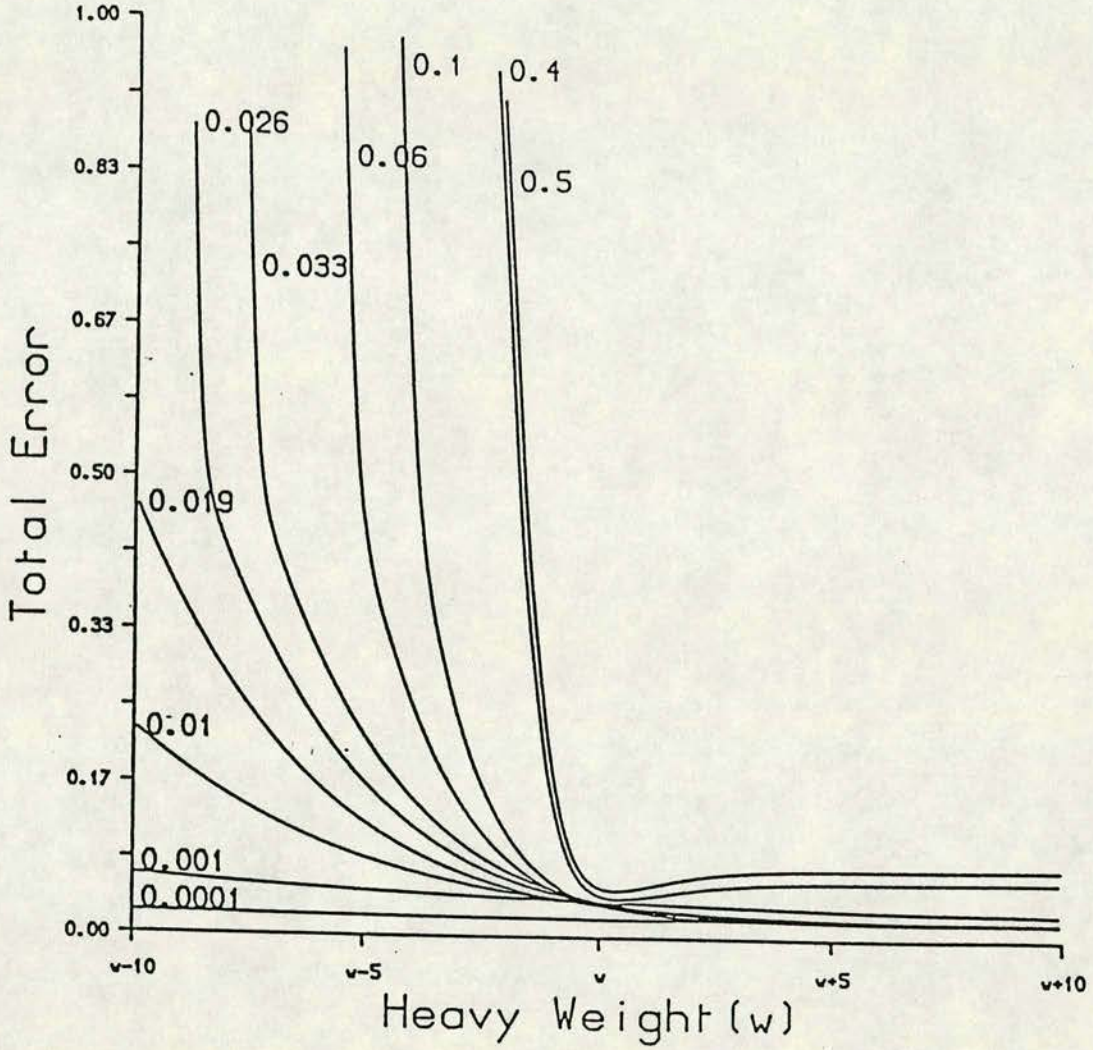

Figure 2.6: Error surface in the direction of a dominant weight for various values of the parameter r.

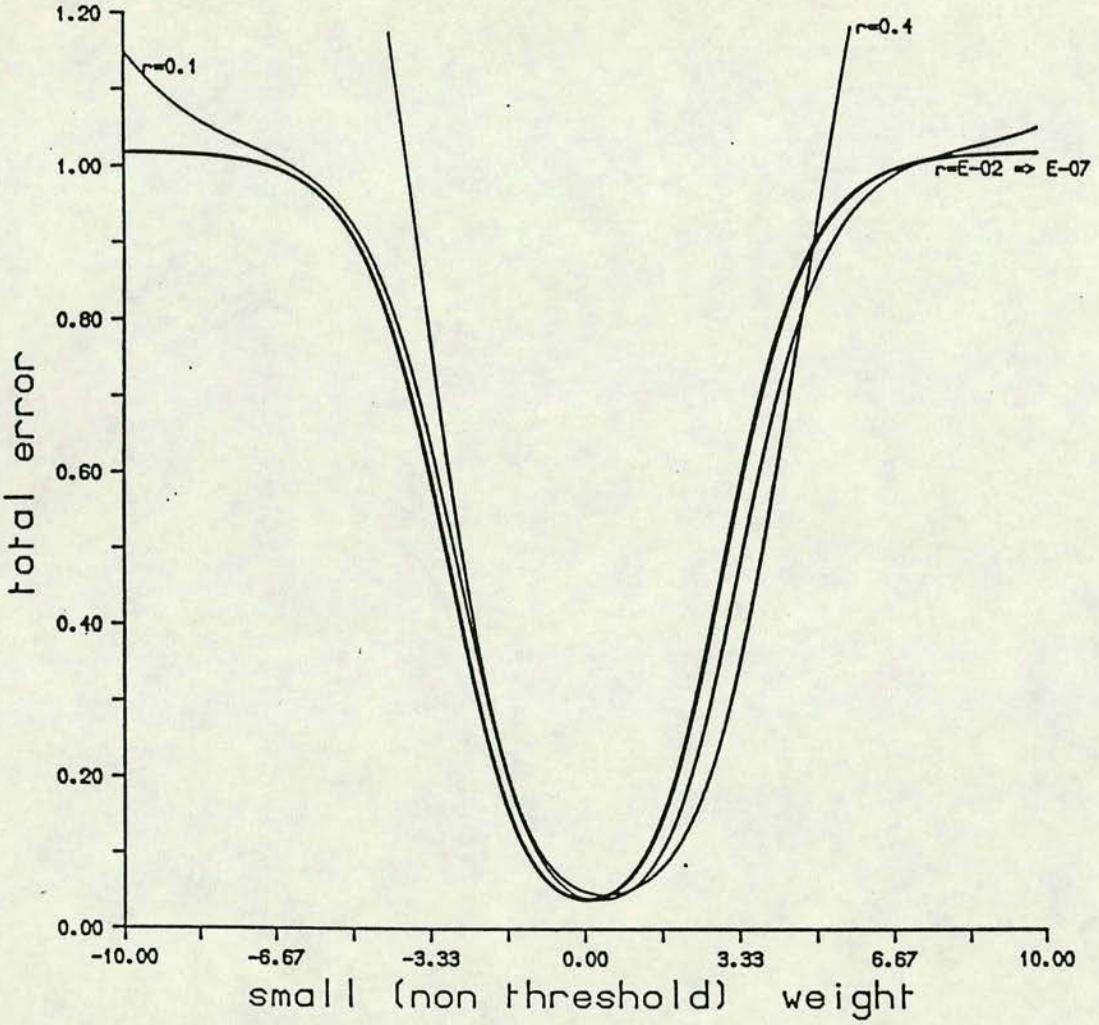

Figure 2.7: The error surface in the direction of a small weight for various values of the parameter r.

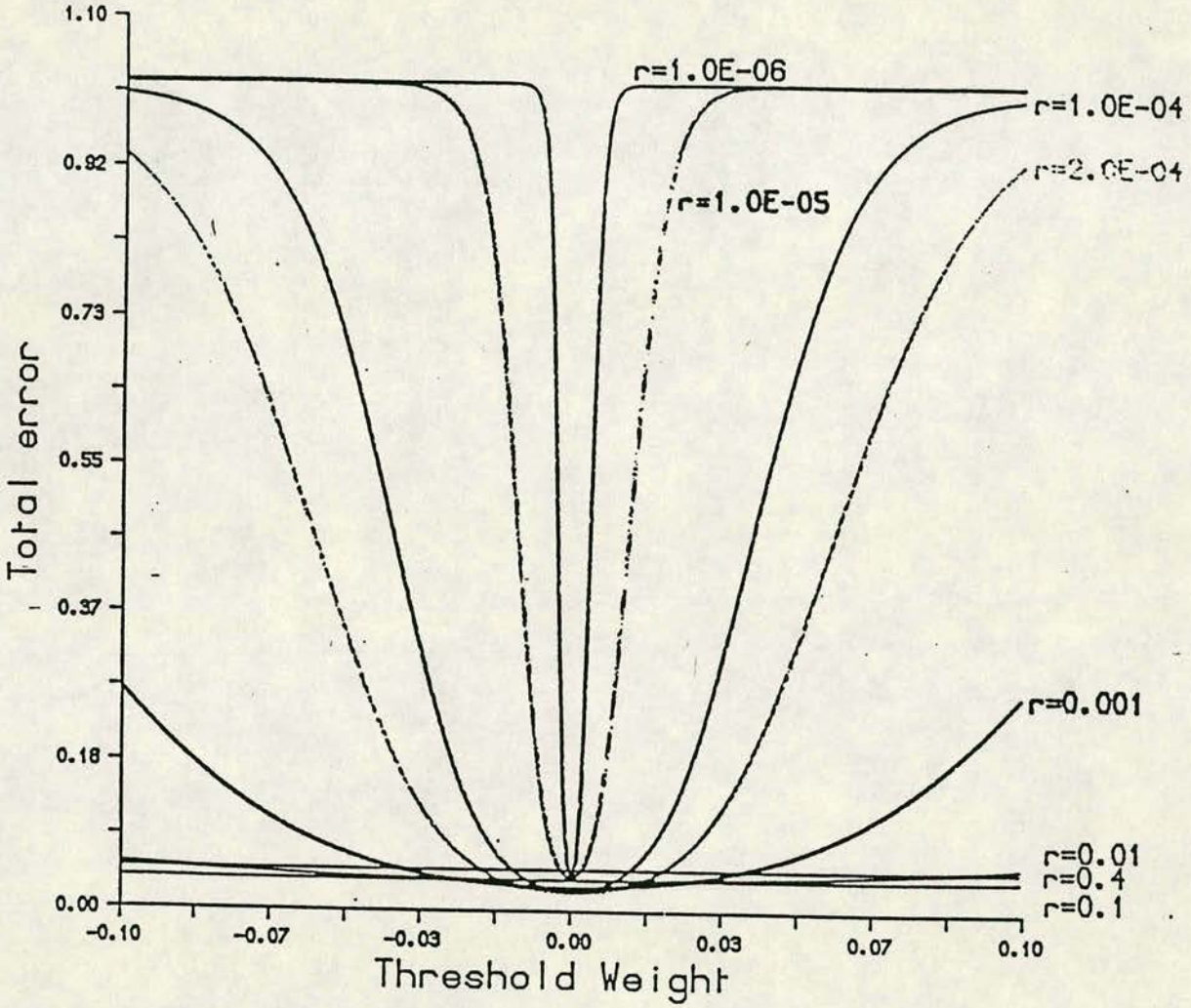

Figure 2.8: The error surface in the direction of a bias for various values of the parameter r.

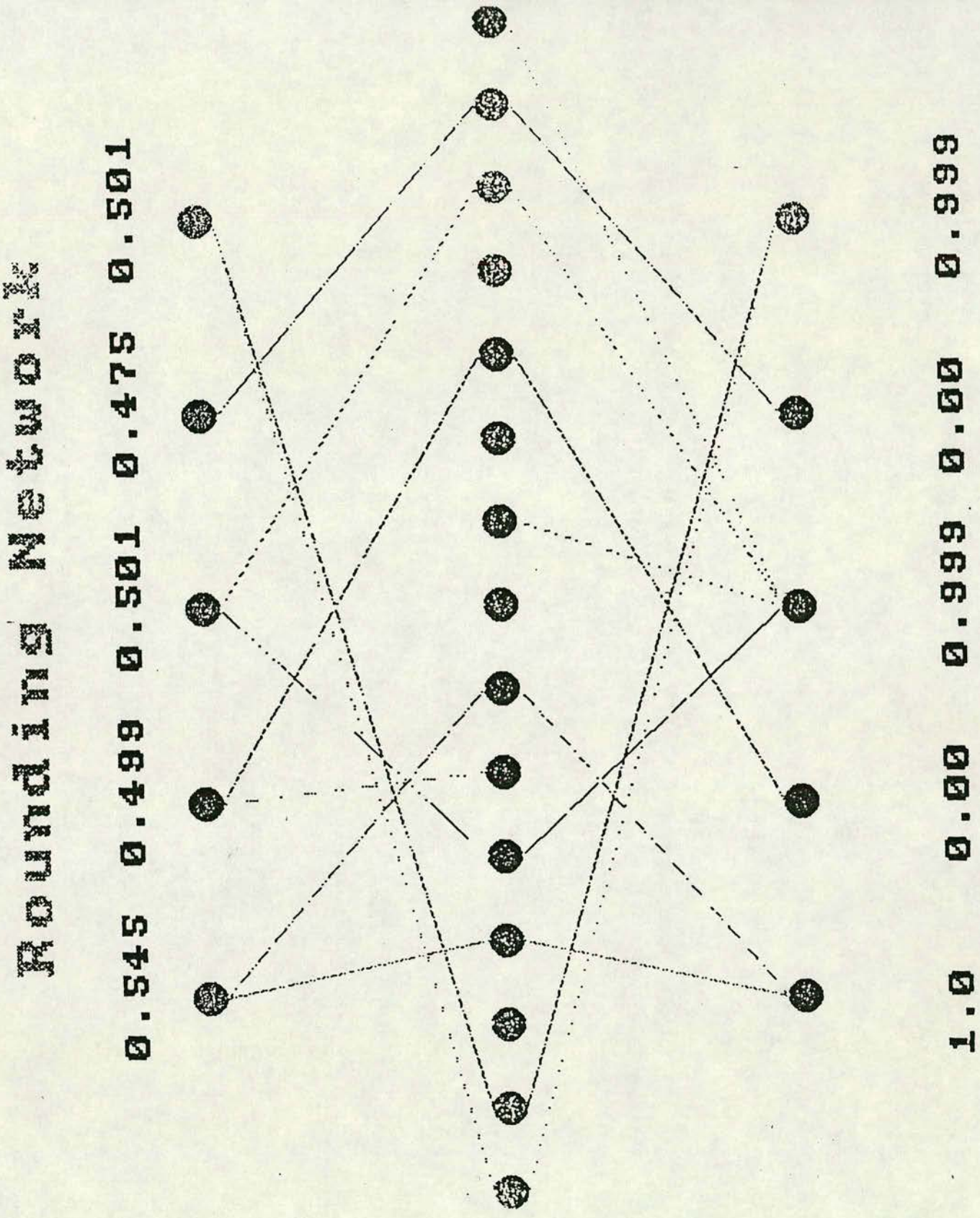

Figure 2.9: Screen dump from a graphics display during the training of a 5-15-5 network. The numbers on the top are the input and the numbers on the bottom are the rounded output. The presence of the dominant weights and redundant nodes can be clearly seen.

becomes a flat plain as would be expected since these weights need to be large.

These maps show that it is the biases that are critical to the stability of the system. Even a small deviation from zero can cause an output node to go totally wrong. When this occurs an output node flips from being near 1 to 0 or vice versa. This was often observed. This is an example of how having the same step size in all directions can cause problems.

As has been stated, there must be at least the same number of hidden nodes as input and output nodes but what is the effect of having more than this? The effect can be seen in figure 2.10. This figure shows the terrain for the first few cycles for 2-N-2 networks with $r = 0.5$. It can be seen from that figure that the terrain for a 2-25-2 network is much steeper than that for a 2-2-2 network. The error surface for the start of training can be seen in figure 2.11 for three difficulties. It can be seen that the gradient is greater for more hidden nodes. An explanation for this can be seen as follows. If it is assumed that at the start of learning each weight is equally important and is changed so as to reduce the error then if the changes are small and

$$\frac{\partial E}{\partial w_{j\alpha_j}^{i\alpha_i}} \propto \delta w_{j\alpha_j}^{i\alpha_i}$$

this then leads to

$$\frac{\mathrm{d}E}{\mathrm{d}w} = \sqrt{\sum_{\{ij\}} (\delta w_{j\alpha_j}^{i\alpha_i})^2}$$

Then if there are N weights in the system

$$\frac{\mathrm{d}E}{\mathrm{d}w} \propto \sqrt{N} \tag{2.3}$$

From the gradients in figure 2.10 this ratio comes to 3.36 and from equation 2.3 the ratio would be expected to be 3.25. (There are 12 and 127 weights in the systems.) Analyses of various systems revealed that hidden nodes frequently had weights of negligible magnitude connected to them and this occurred more frequently the greater the number of nodes. These 'redundant' nodes can be seen clearly in figure 2.9.

The influence of the momentum parameter can be very important in finding a solution. For example with a 7-25-7 network and $r = 0.5$ a solution was found in only six cycles when there was no momentum ($\alpha = 0$), whereas the same system with $\alpha = 0.6$ did not find a solution until more than 50 cycles. The descent during

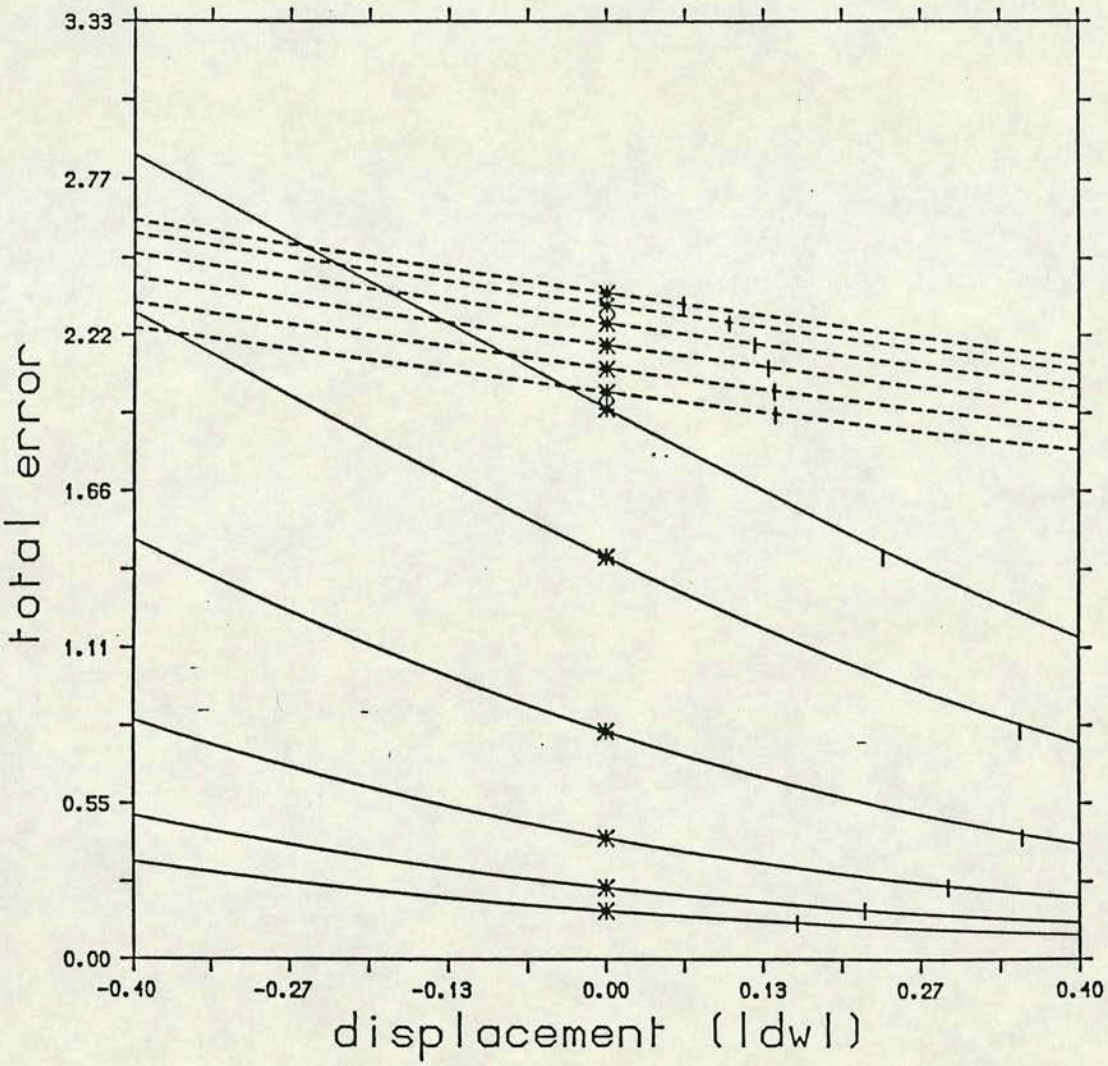

Figure 2.10: The terrain for the first six cycles of descent in a 2-2-2 and 2-25-2 system. The dashed lines are for the smaller network.

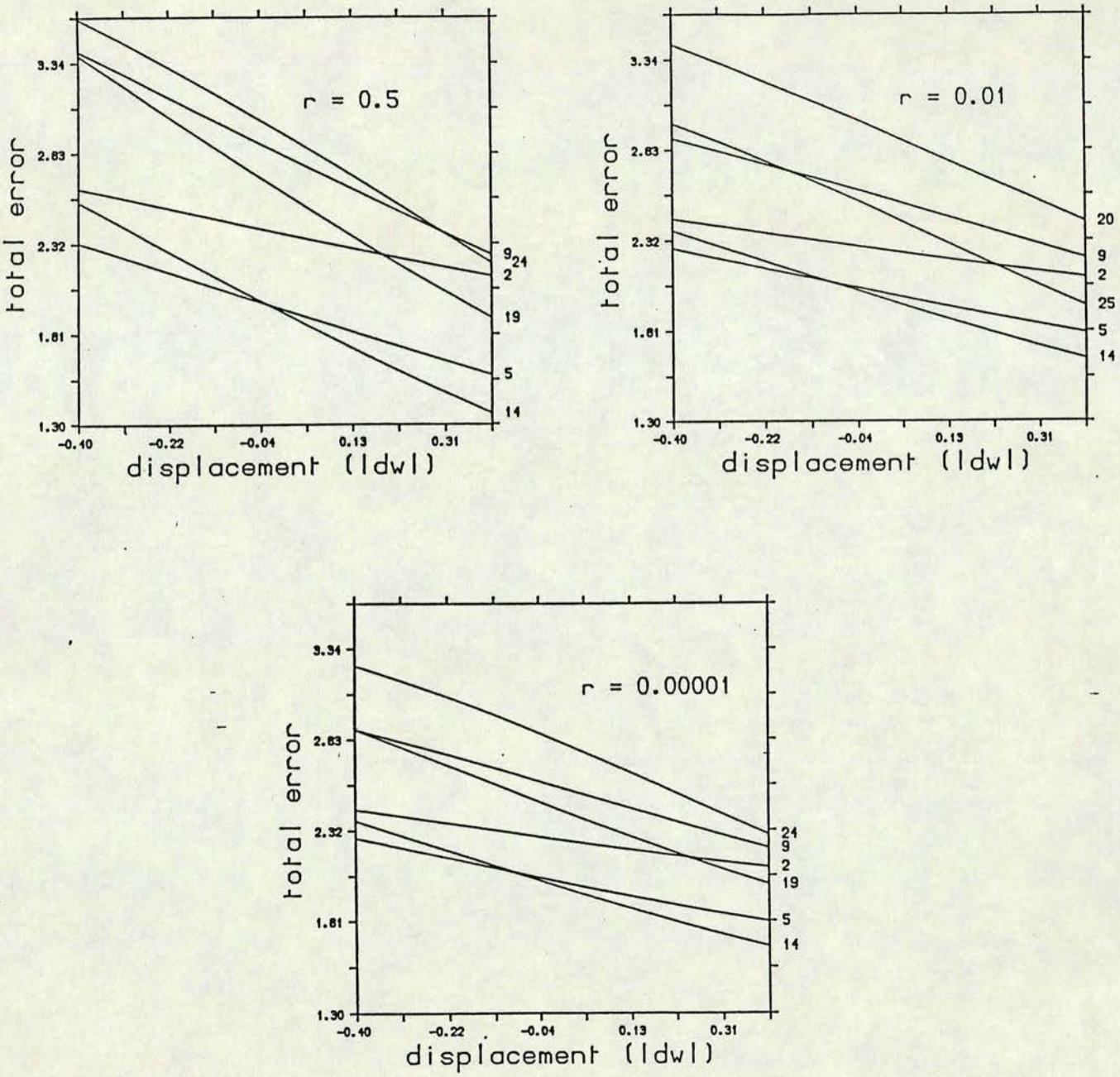

Figure 2.11: The terrain at the start of learning for the 2-N-2 network for three values of r. The number of hidden nodes is shown on the right of the figures.

the initial few cycles is very similar in the two systems but in the second case the system landed on a plateau and took more than forty cycles to get off it when a cliff like edge was found. This terrain can be seen in figure 2.12. It appears that by chance that the system with no momentum avoided this plateau but that with momentum landed on it. It is this kind of unpredictable behaviour that the annealing described below is an attempt to avoid.

Figures 2.13 show the scaling of the learning time for $r = 0.5$. These graphs show the total time to solution for each system averaged over 50 - 100 runs with different random starts. They show clearly that the total number of cycles to learn decreases with the number of hidden nodes and that the widths of the distributions decrease. If the total learning time is plotted against the log of the difference between the number of input and hidden nodes then (see figure 2.14), at least for small numbers of hidden nodes, the curves appear to be linear. This suggests that the learning time varies as

$$t \sim f(n_1) \log(n_2 - n_1)$$

where n_2 is the number of hidden nodes. Variation from this rule for large numbers of nodes may be due to the appearance of the unused nodes.

The learning algorithm failed to find solutions for values of $r < 0.0075$. Solutions could sometimes be found if the parameters η and α were adjusted interactively but this is not very satisfactory as the form of the error surface cannot be predicted in advance. It was also found that the smaller r the more runs failed to converge and the wider the distribution as can be seen from figures 2.1 and 2.2.

2.4 Annealing

It has been shown that it is much easier to train a network to solve a problem with large values of r than for small. Since r can be varied continuously, it should be possible to train the network on the largest value and slowly reduce it. To do this the effect on the error at an output of changing r is needed. Thus the value of

$$\frac{\partial e^p_{\alpha_L}}{\partial r} = \sum_{\alpha_{L-1}} \frac{\partial e^p_{\alpha_L}}{\partial v^p_{L-1\alpha_{L-1}}} \frac{\partial v^p_{L-1\alpha_{L-1}}}{\partial r}$$

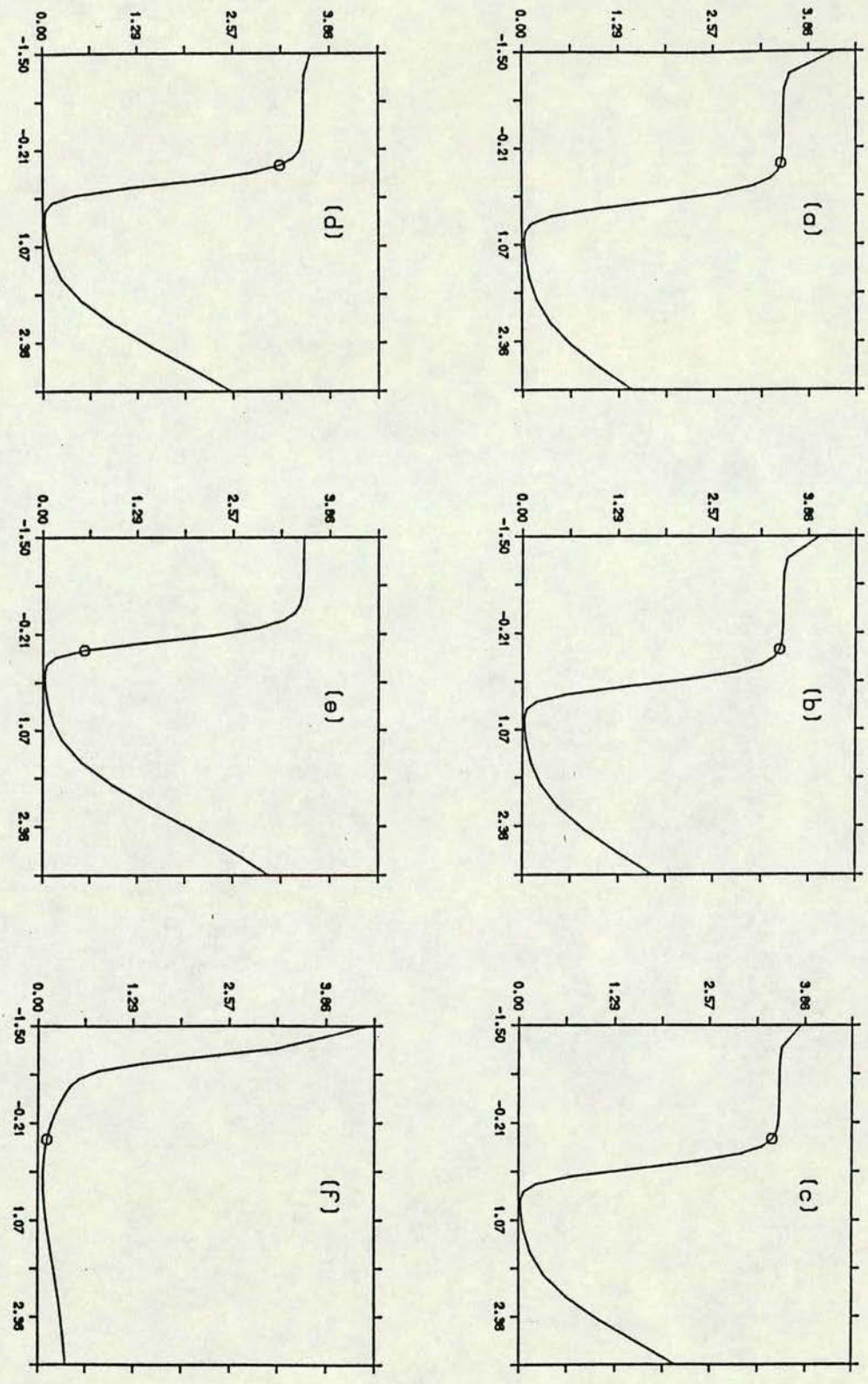

Figure 2.12: Location of a cliff-like edge in the gradient descent of the error surface of a 7-25-7 system with $\alpha = 0.6$. The figures show the progress of the network (the circle) as it comes to the end of a long, slow plateau. No such problem was encountered when α was set to zero.

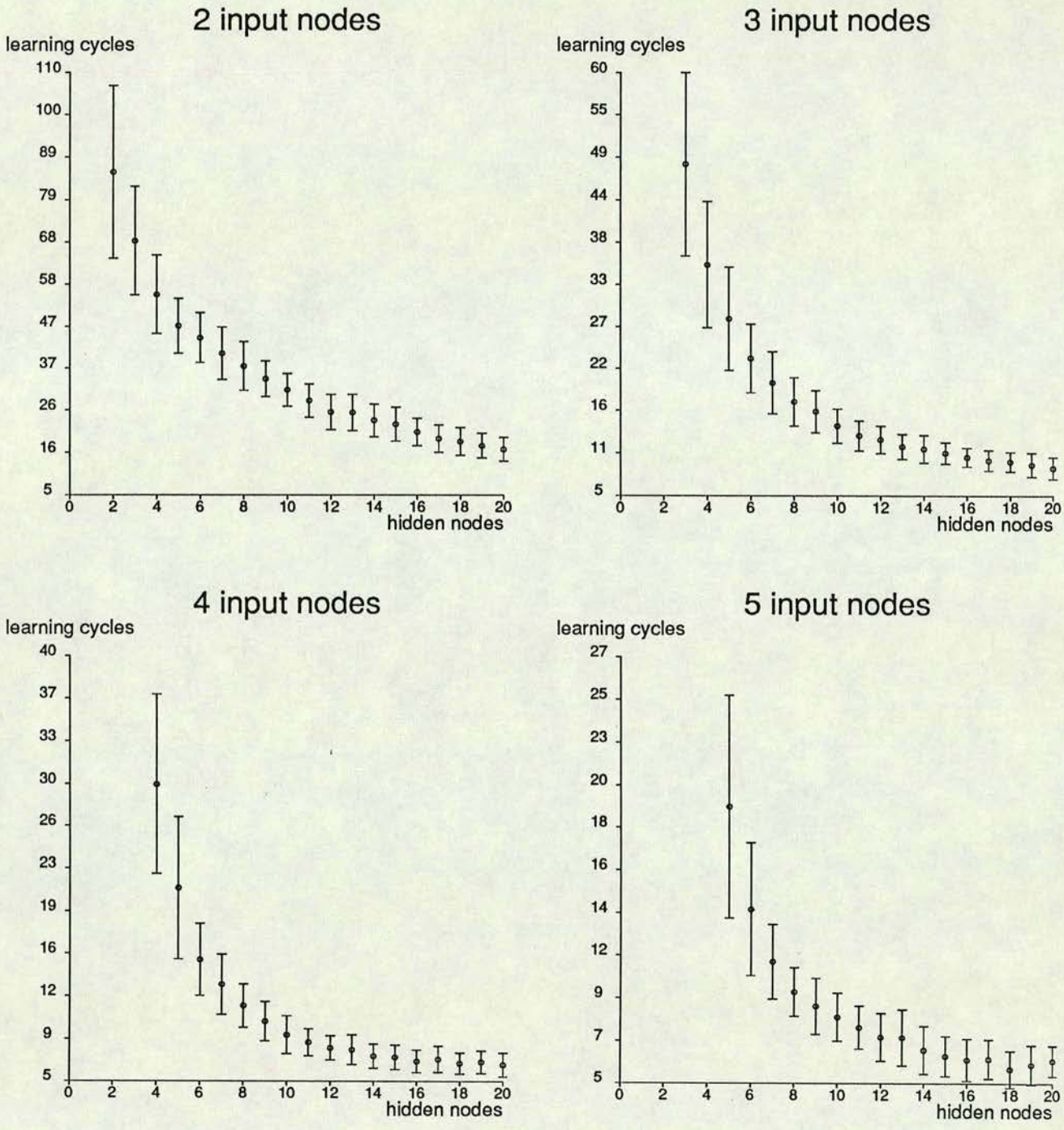

Figure 2.13: The effect on learning time of the number of hidden nodes.

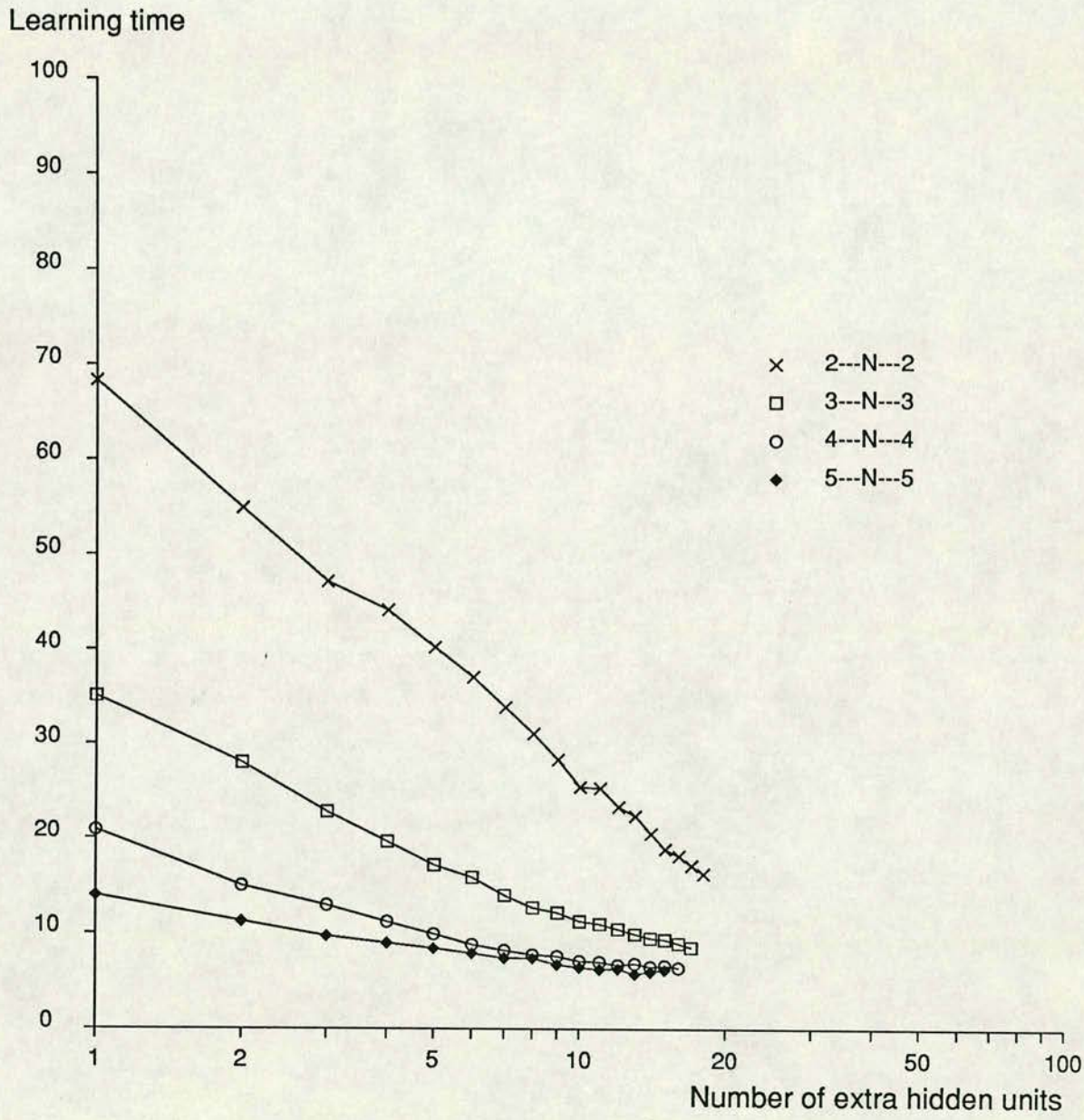

Figure 2.14: The scaling of the learning time with the number of excess hidden nodes.

is needed. Because the input patterns are a function of r, the node states are now given by

$$v^p_{i\alpha_i} = \begin{cases} \mathcal{V}(\phi^p_i, \beta), & \text{if } i > 1, \\ P(p, r, \alpha_i), & \text{if } i = 1. \end{cases}$$

where P is the pattern annealing function. By the chain rule this can be expanded

$$\frac{\partial e^p_{\alpha_L}}{\partial r} = \sum_{\alpha_{L-1}} \frac{\partial e^p_{\alpha_L}}{\partial v^p_{L-1\alpha_{L-1}}} \sum_{\alpha_{L-2}} \frac{\partial v^p_{L-1\alpha_{L-1}}}{\partial v^p_{L-2\alpha_{L-2}}} \cdots \sum_{\alpha_1} \frac{\partial v^p_{2\alpha_2}}{\partial v^p_{1\alpha_1}} \frac{\partial v^p_{1\alpha_1}}{\partial r}$$

$$\frac{\partial e^p_{\alpha_L}}{\partial v^p_{L-1\alpha_{L-1}}} = \frac{\partial e^p_{\alpha_L}}{\partial v^p_{L\alpha_L}} \frac{\partial v^p_{L\alpha_L}}{\partial \phi^p_{L\alpha_L}} \frac{\partial \phi^p_{LlnL}}{\partial v^p_{L-1\alpha_{L-1}}}$$

For $l > 1$

$$\frac{\partial v^p_{L\alpha_L}}{\partial v^p_{L-1\alpha_{L-1}}} = \frac{\partial v^P_{L\alpha_L}}{\partial \phi^P_{L\alpha_L}} \frac{\partial \phi^p_{L\alpha_L}}{\partial v^p_{L-1\alpha_{L-1}}}$$

This then gives

$$\frac{\partial e^p_{\alpha_L}}{\partial r} = \frac{\partial e^p_{\alpha_L}}{\partial v^p_{L\alpha_L}} \frac{\partial v^p_{L\alpha_L}}{\partial \phi^p_{L\alpha_L}} \sum_{\alpha_{L-1}} \frac{\partial \phi^p_{L\alpha_L}}{\partial v^p_{L-1\alpha_{L-1}}} \cdots \sum_{\alpha_1} \frac{\partial \phi^p_{2\alpha_2}}{\partial v^p_{1\alpha_1}} \frac{P(p, r, \alpha_1)}{\partial r} \tag{2.4}$$

The amount by which a change in the the value of r will affect the error can be estimated

$$\delta e^p_{\alpha_L} = \frac{\partial e^p_{\alpha_L}}{\partial r} \delta r \tag{2.5}$$

The error at any output node must not increase so much as to cause the system to leave the valley. To do this the output node with the largest error is found. This is then used to calculate the change in r which will change the error by a given amount

$$\delta r = \frac{\delta E_j}{\frac{\partial \sum_p e^p_j}{\partial r}} \tag{2.6}$$

The learning scheme now becomes

1. Set r to its starting value of r_0. In this case $r_0 = 0.5$.

2. Learn to tolerance with the patterns for the current value of r.

3. Calculate the change in r from the allowable increase in error. This increase must be such that a few learning cycles are needed and the increase is not such that the valley is left.

4. Go back to 2.

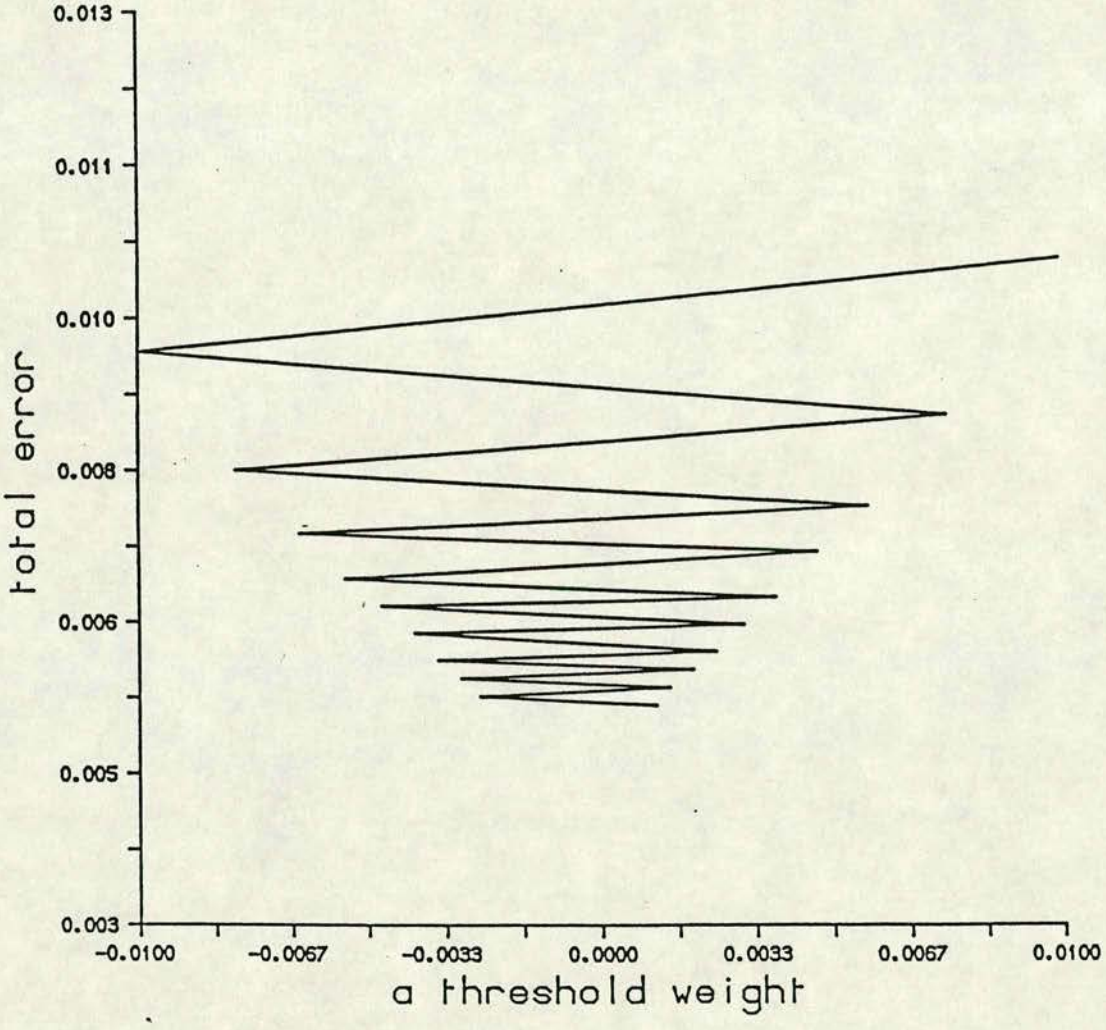

Figure 2.15: Ascent from an error minimum using the gradient descent algorithm.

This method proved very successful in allowing the network to solve tasks down to $r = 0.0001$. However it could be seen that the updating procedure became increasingly inefficient as r decreased. To reach even lower values of r it was necessary to improve the weight updating. If r was pushed down too far the flipping described earlier occurred. Even if the system did find the crevasse-like minimum of the biases it would have great difficulty staying in it. Figure 2.15 shows how the system can climb up a valley using the gradient descent by bouncing from one side to another. This effect, which ultimately leads to the system hanging on the flat region outside the valley can be explained by the step size at the lowest point being just too big to take the system down the valley, instead it ends up higher up on the opposite side. The annealing results in the narrowing of the valley which prevents the momentum strategy from working. The onset of this behaviour could be detected when two successive weight changes are in opposite directions and the latter is of greater magnitude. When this occurs η is reduced for that weight. The reducing schedule must be one that takes into account by how much the step size is too big. For example, it would not be adequate to multiply by some constant factor. The momentum which is important when the gradients are small or in valleys when the step size is small must not be used when the system crosses a valley bottom. So the method used to control valley descent was as follows

- If the calculated weight change and the previous weight change have the same sign then the system is on a slope so update the weights as usual otherwise

- If the current weight change is greater in magnitude than the previous change then

$$\eta_{kl}^{ij} \mapsto \left| \frac{\Delta w_{kl}^{ij}(n-1)}{\Delta w_{kl}^{ij}(n)} \right| \eta_{kl}^{ij}$$

Using these techniques it was found that r could be reduced to 10^{-8} (the only limit found was the precision of the computer used for the simulations). The learning time down to $r = 0.005$ can be seen from figure 2.16. From this figure it can be seen that up to about $r = 0.075$ annealing actually slows the learning down as would be expected since the ordinary algorithm can cope. This indicates that

the annealing is not needed until about this value. After this value the learning time increases far more slowly. From figure 2.2 it can clearly be seen that the improved algorithm outlined here is much more stable than the normal algorithm even with the new potential. All of the runs converged compared to 47% in the case of the normal algorithm. So even though as can be seen from the figure, the annealing does not greatly decrease the convergence time if the conventional algorithm does converge, it converges far more consistently.

2.5 Conclusion

This simple problem has revealed many of the difficulties encountered with back-propagation and ways of improving the algorithm. It has shown that judicious choosing of functions can improve learning times and that in this case controlling the patterns can allow learning to proceed further. Studying the error surface has explained why controlling the step size at the end can allow the system to solve harder problems and allow momentum to be used more effectively. It has for example explained why the same step size is not suitable for all dimensions of the error surface due to the fact that they may have very different forms. There are many ways of controlling the value of the step size and a summary can be found in Tollenaere(1989).

More work using the annealing technique outlined above has been carried out by Smieja(1989) and was again shown to greatly improve on the basic algorithm. In this work the network was trained to recognise patterns when presented with noisy examples. Smieja found that the network performed much better if it was trained on increasingly noisy patterns rather than on the most noisy patterns first of all.

In summary, this chapter has shown that using some of the known facts about the problem to control learning can greatly improve the performance of the network. It is however rather disappointing that such an improvement in performance is achieved by explicitly writing in the significance of 0.5.

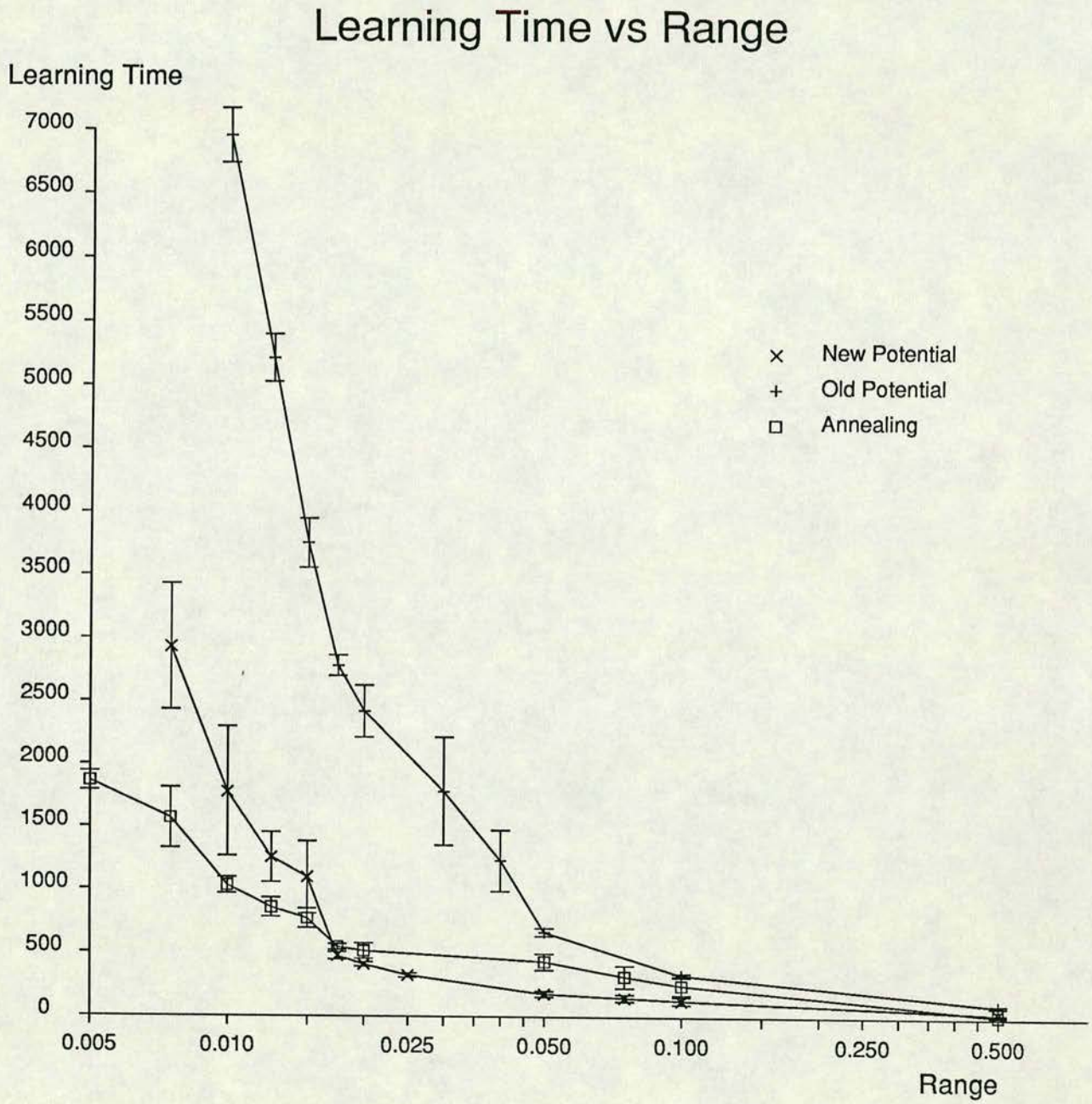

Figure 2.16: The length of time the improved an basic algorithms took to learn to tolerance 0.1 for for various values of r.

Chapter 3

Implementation

One of the major problems encountered in the investigation of layered networks is that of the time each simulation can take. Even networks as small as tens of nodes a layer may take many thousands of iterations to converge to a satisfactory tolerance. When a network is made larger the error surface tends to become more complex and so convergence takes longer and each iteration is made slower by the fact that the number of calculations required increases as the square of the number of nodes. Since in the work carried out in this thesis it was desired to simulate fairly large networks (the largest having four hundred input nodes) it was felt necessary to write a simulator capable of doing this in a reasonable time.

In Edinburgh there is a multi-user transputer based machine built by Meiko limited [1] and all the simulations carried out in the following chapters were performed on this. A description of these facilities can be found in EUCS(1989). The transputer (INMOS 1987) is a 32 bit CMOS microcomputer which has four communications links which allow processors to communicate with each other. The Computing Surface in Edinburgh has about four-hundred such processors that have been arranged so that there are many single user seats (called domains) ranging in size from 133 processors down to a single processor for editing. At present four of these domains have high resolution graphics capabilities. The transputer instruction set achieves efficient implementation of high level languages and provides direct support for the occam (INMOS 1988) model of concurrency. Occam is based on the process model of concurrency. A process is an independent computation

[1] Meiko Scientific, Aztec West, Bristol

which has its own program and data and can communicate with other simultaneous processes. In occam this communication is achieved via one-way channels linking the two processes explicitly defined in these processes. Processes can be interconnected by such channels to form larger, more complex, processes. This is illustrated in figure 3.1. In this view of concurrency there are two ways that

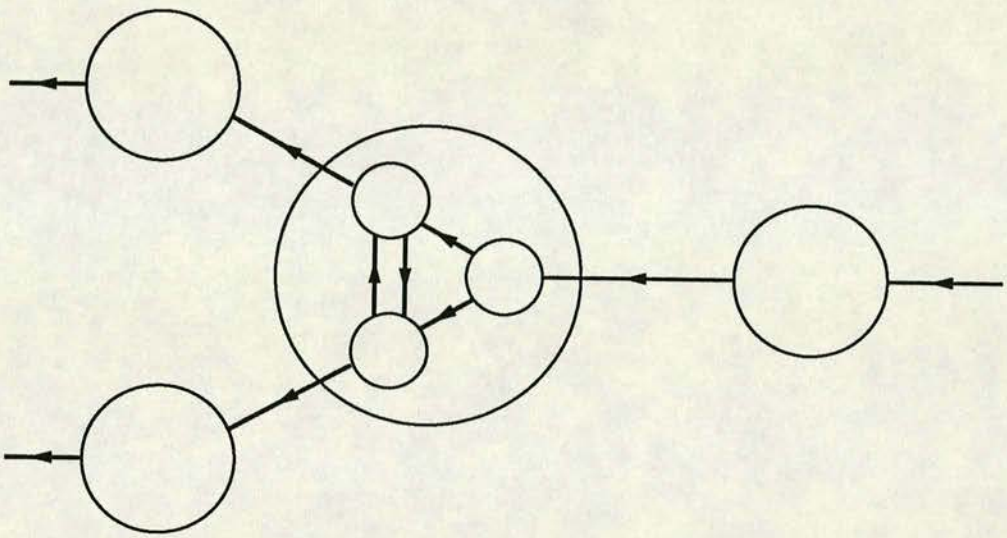

Figure 3.1: Here is shown how small processes (circles) can be combined using channels (arrows) to form larger processes.

the communications can be controlled. When independent processes are communicating the communications must be synchronised. It is possible that a process has more than one input or output channel from which or down which it could receive or send a message. In this situation there needs to be a way of choosing one of these options. In occam there is a construct which enables this choice to be made on input. It is possible to do this for output, though harder to implement, but if there are multiple processors the option can not exist for both.

The transputer suits well this view of concurrency by supporting the idea of many processes. The links which allow the processors to communicate mean that once a problem has been reduced to smaller processes or sub-problems these tasks can then be distributed over the processors with the hard links acting as channels.

3.1 The Method Used

The transputer is a so called MIMD[2] device since all the processors run separate programs on their own data. There are other types of parallel machine; for example the SIMD[3] type which run the same program on different data. Different problems are suited to different machines. There are two main ways of dividing a problem on an MIMD machine: algebraic and geometric decomposition. In the first case the algorithm used in the problem is reduced to parts that can be performed independently and in the second case the problem itself is divided. For example, in weather simulations the map of the world could be divided between the processors so that they would communicate when something moved from the part of the world controlled by one processor to that controlled by another. This is geometric decomposition. In other situations a problem can be broken down into independent parts that can be performed in parallel. For example, if many analyses of the same object were to be formed in parallel this would be algebraic decomposition.

One of the main considerations when choosing a method is the amount of inter-processor communications involved. It is for example very undesirable for the whole simulation to have to come to a complete halt too frequently and it is desirable to have most of the communications between nearby processors so that there is little need for the routing of messages.

There are many ways of so dividing back propagation and some of these will be discussed later in this chapter. In the case of this simulator it was by way of division of the matrix-vector multiplications. This is a geometric decomposition.

In both the forward and back passes of the learning process matrix-vector multiplications are performed. For example, in the forward pass the calculation of the potential at a node from the states of the nodes in the layer above (equation 1.13) is just a matrix-vector multiplication. The interconnecting weights form the weight matrix and the states of the nodes above form the vector. The type of network considered here only allows connections between adjacent layers (i.e.

[2]Multiple instruction multiple data-stream

[3]Single instruction multiple data-stream

links cannot skip layers), which means that for an n-layer network there are $n-1$ weight matrices.

This form of parallelism is performed by dividing the elements of the weight matrices amongst the processors. Each processor then does a partial matrix vector multiplication and produces a contribution to the final vector. Summing these vectors will produce the result: in the case of the forward pass this is the potential vector ϕ_l^p. As was said above matrix-vector multiplications are performed in the back pass. There is a complication as can be seen from equation 1.17. Here there is in fact a vector-matrix multiplication and so the transpose of the weight matrix is needed. This causes problems in the distribution. For example, if the matrices were sliced, that is each processor had some of the rows, all the weights needed to calculate some of the elements of the resultant vector would be on one processor for the forward pass but some would be on each of the processors in the case of the back pass. One solution is not to assume slicing, neither vertically nor horizontally, but assume each processor has elements from potentially any part of the weight matrix. Each processor would then calculate a contribution to each element of the vector. These complete vectors of partial elements would then be communicated and summed to produce the entire vector.

There is another advantage with this method in that partial connectivity can be easily implemented. If this is the case the weight matrix will be sparse and if it is very sparse just representing the non-existent weights as zeros in the matrix becomes very inefficient in that it gives no speed up. If only the weights that exist are stored then there is the possibility that some of the processors would have significantly more calculations to do than others: there would not be load balancing. In the method chosen matrix elements can be divided equally amongst the processors giving a reduction in computation time the greater the sparsity of the connectivity.

For a small number of processors (up to about 64) a torus (see figure 3.2) is a fairly efficient configuration (Radcliffe 1989). Its main advantage and the reason for its popularity is the ease of the communications. The problem here is how do all the processors communicate their part of the result to all the others? This must be achieved as quickly as possible so that the next stage in the calculation

can be performed. In this implementation a processor carries out its part of the matrix-vector multiplication, whether it is for the forward or back pass, and places the result in a communications block. Once a pre-set number of these have been performed the message block is sent out in both directions in one dimension of the torus while receiving the result of the other processors. The processor then has the total of the row which is then sent in both directions vertically. Once the processor has received all the horizontal and vertical messages and calculated its own it has the final vector. One advantage with this method is that since all the patterns are independent the next block of multiplications can be performed whilst the communications are underway with the previous block: calculations are being made continuously [4]. If the torus has m rows and n columns then for each of the processors there will be $m-1$ messages received horizontally and $n-1$ vertically. This then gives, including its own messages, a total of $m+n$ communications. If there are p processors this is a minimum for $m = n = p^{\frac{1}{2}}$. That is the program is more efficient if the grid is square. For a p processor torus simulating a network with N neurons per layer one matrix-vector multiplication involves a total of 2 message sends and $2(p^{\frac{1}{2}}-1)$ receives, giving a total of $2p^{\frac{1}{2}}$ communications. The size of a message is $\approx N$ giving a communications load for each processor should that scale as $2Np^{\frac{1}{2}}$. This means that if more processors are used the communications time will increase and will eventually win over the calculations which vary as N^2p^{-1}. There will therefore be an optimal number of processors for any particular problem size.

As was said above, this implementation is most efficient for partially connected networks (not all the possible inter-layer connections are present). This effect along with the effect of communications time scaling can be seen in figure 3.3 which shows how the program speeds up as the connectivity varies from fully connected for various sizes of network. The times given are fractions of the time taken for the fully connected network to do one cycle with 100 training patterns. All the runs in this figure were done on a sixteen processor torus and it can clearly be seen that the program is much more efficient for larger networks. For example, a 40-40-40 network with 10% connectivity takes about 82% of the time of the fully

[4]This overlapping of communications has not yet been fully implemented.

connected network whereas the 200-200-200 network takes about 40% of the time. For small networks there are much better ways of distributing the problem some of which will be discussed later in this chapter.

Since the patterns are not being distributed it would seem reasonable to assume that the speed of the simulation would be linear with the number of training patterns. Figure 3.4 shows how the length of time a network with forty nodes in a layer running on sixteen transputers takes to do one iteration against the size of the training set. As can be seen it is indeed linear. A least squares fit on the data gives a gradient of 0.0144 ± 0.0001 and an intercept on the y axis of 0.0104 ± 0.124. So the scaling is clearly linear and the overheads are negligible.

A far more interesting scaling is in the number of nodes in a layer. From the above estimates of the scaling of the communications and calculations time, it may be anticipated that the length of time the program will take to perform one iteration for a given number of patterns will vary as

$$t \sim \frac{2bN(N+1)}{p} + 2Ncp^{\frac{1}{2}} + a$$

where the first term is communications time and the second is calculations time. This gives the variation with the number of nodes as

$$t \sim \frac{2b}{p}N^2 + \left(\frac{2b}{p} + 2p^{\frac{1}{2}}c\right)N + a$$

Where b and c are constants and a is a term accounting for other dependencies such as the length of time it takes a message to get around the system. It would be expected that the time taken per iteration would vary as second order polynomial. Figure 3.5 shows this scaling and a second order polynomial fit to the data gives $a = 0.148$, $b = 0.0014$ and $c = 0.0032$.

As was said above the communications time of this program is dependent upon the number of processors. It would therefore be expected that the speed of the program would not scale very well with the number of processors. Table 3.1 shows the time taken to perform one iteration for three fully connected network sizes each with one hundred training patterns for various numbers of processors. In each case the processors were arranged in a torus. It can be seen that indeed the scaling is very poor especially (as would be expected since the calculation time is

the least) for the small network. The worst aspect is that even for small numbers of processors the program actually runs more slowly. For example, for a network with forty nodes per layer, the sixteen processor configuration runs at 1.59 seconds per iteration whereas that with nine processors runs at 1.43 seconds per iteration. In any problem if enough processors are used this will happen but in this case because the communications time is dependent upon the number of processors it happens for a small number of processors. From this table it can be seen that for most problems the optimal number of processors is about twenty-five. Using these timings estimates for the values of a, b and c can be made. As can be seen from the above, the variation is small and the unaccounted for time is only about 12% of the total.

From table 3.1 and equations 1.16 - 1.18 and 1.11 - 1.14 an estimate can be made of the number of floating point calculations made by the program each second. This implementation uses a look-up table for the activation function (equation 1.12) and so those calculations can be omitted. It can be shown that the number of calculations in a single iteration (one pattern presentation) varies as

$$n \sim 14N^2 + 15N$$

Table 3.3 shows how this number of calculations (generally called flops) varies with the size of the problem and the number of processors. The T800 floating point transputer can run at about $1Mflop$ and as can be seen from this table, the best performance reached is only about half this. This table shows that the communications system used greatly reduces the performance of the program: for large networks the speed could be doubled and for small networks it could go more than ten times faster.

3.2 Capabilities

As has been shown above, this simulator is most efficient for large networks; this is indeed the purpose for which it was designed. It was in fact designed to simulate large partially connected networks and it has been shown that in this case it scales well. The present version allows the user to specify the connectivity in

four ways. The first way is by specifying the number of weights present and they are then distributed at random amongst those possible. Secondly a probability is specified and each weight then has that probability of existing. The third way is by setting up connections as if the two layers are lattices and n^{th} nearest neighbour connections are set up. The final method is by setting up windows of connections. For example, a simple window would be that only the weights connecting a neuron to the three neurons below it are used (see fig 3.6). Future implementations will read connectivity specifications from file so that if the connectivity desired cannot be specified using combinations of the above the actual connections desired could be put in file. The user can have a different type of connectivity, different values of the learning parameters etc for different layers. The main functions of back-propagation are contained within libraries to allow the user to change them with a minimum of effort. Most runtime events can be logged and command files can be used to alter them during learning. A full specification of the capabilities of the program can be found in Richards(1989).

3.3 Other Methods

In the case of neural networks there are three obvious forms of parallelism that can be exploited, namely division of the network, division of the training set and division of the matrix-vector multiplications. Each method has its advantages and its disadvantages.

The first method, as described in Beynon(1987), is the one that is prehaps the most intuitive. In this method, each processor has a subset of the neurons in the network. So if there are n_i neurons in layer i and p processors, then each processor is allocated approximately n_i/p neurons. Each processor stores the values of the weights going to its neurons. A processor must send the state of its neurons and receive the states of the others. Each message packet sent by a processor visits each of the other processors once. Each processor therefore sends one message and receives $p-1$ each of length of n_i/p. This gives the total communications load for each processor of n_i for that layer. With this system the communications load is independent of the number of processors but at the cost of a more complex

communications system.

The second method is the simplest. The final gradient array is just the sum of the individual arrays for each of the patterns in the training set. The patterns can therefore be divided amongst the processors and the gradient arrays summed before the weights are updated. This method has the advantage that communications need only take place after all patterns have been presented. This is therefore most efficient with batch update. A problem is that the whole network must be stored on each processor which means smaller networks can be simulated than with weight distribution.

Another method of distributing the weights which scales better is to divide the processors into two groups: one doing the forward pass and the other doing the back pass. Doing it this way, in the forward pass group the rows of the matrices can be distributed and in the other group the columns can be distributed. In either case each processor would know the states of all neurons in the network. Once a processor has calculated the states of its neurons these must be sent and it must receive the states of the other neurons. This amounts to sending one message and receiving $p-1$. The size of a message is approximately the number of rows allocated i.e. N/p. The communications load therefore scales as N as in the first example.

3.4 Conclusion

The program described in this chapter was the first parallel program written by the author and so in hindsight many of the methods used would be changed. For example the toroidal configuration was used for the ease of the communications system used; this is a daunting problem when a person first encounters parallel programming and is the cause of most problems. In light of experience a more efficient configuration would be used since for problems where global broadcasting (which is what is needed here) or where any processor could potentially need to communicate with any other a torus rapidly becomes very inefficient. This is discussed in detail in Radcliffe(1989) and the properties of random configurations are discussed. There are configurations that can be derived from graph theory

(Beynon 1987) which have near minimal average and maximum inter-processor distances which would be far more efficient for large numbers of processors, the trade off being that the routing problems are far greater. These problems can be solved with generalised communications harnesses (Clarke 1989) which for certain types of communications problems work far more efficiently than anything that could be written in occam. (This particular harness uses the transputer assembly language.)

The strategy chosen for the distribution of the algorithm has its drawbacks as well. The main one being that the communications time is dependent on the number of processors. There are other (more complex) ways of doing such as those described above which (if the size of the problem is big enough) are independent of this number. This has obvious advantages: the greater the number of processors the faster the simulation. However, this simulator is a working, flexible piece of code that has (and is) been used by several people and was used for the simulations in the following chapters.

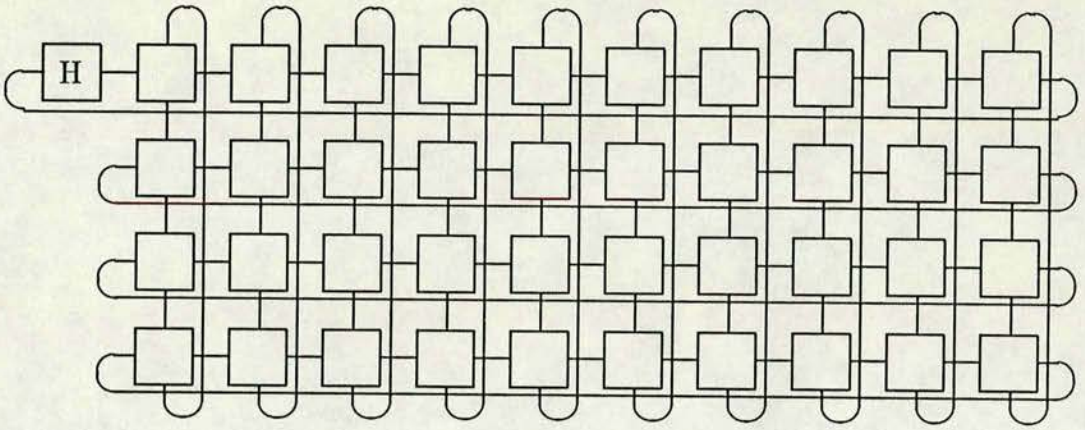

Figure 3.2: An example of a toroidal configuration.

p	40	100	200
4	1.90	8.70	31.0
9	1.43	5.20	16.3
16	1.59	4.77	12.7
25	1.60	4.43	10.9
36	1.94	5.02	11.3
49	1.98	5.08	11.5

Table 3.1: The time taken for three networks with different numbers of nodes in each layer to perform one iteration for various (p) numbers of processors.

p	a	b	c
4	0.108	1.37×10^{-3}	4.17×10^{-3}
9	0.120	1.35×10^{-3}	3.40×10^{-3}
16	0.127	1.32×10^{-3}	3.73×10^{-3}
25	0.152	1.34×10^{-3}	3.17×10^{-3}
36	0.172	1.29×10^{-3}	3.44×10^{-3}
49	0.227	1.92×10^{-3}	2.90×10^{-3}

Table 3.2: Estimates of the coefficients for the time taken to do one iteration for various numbers of processors.

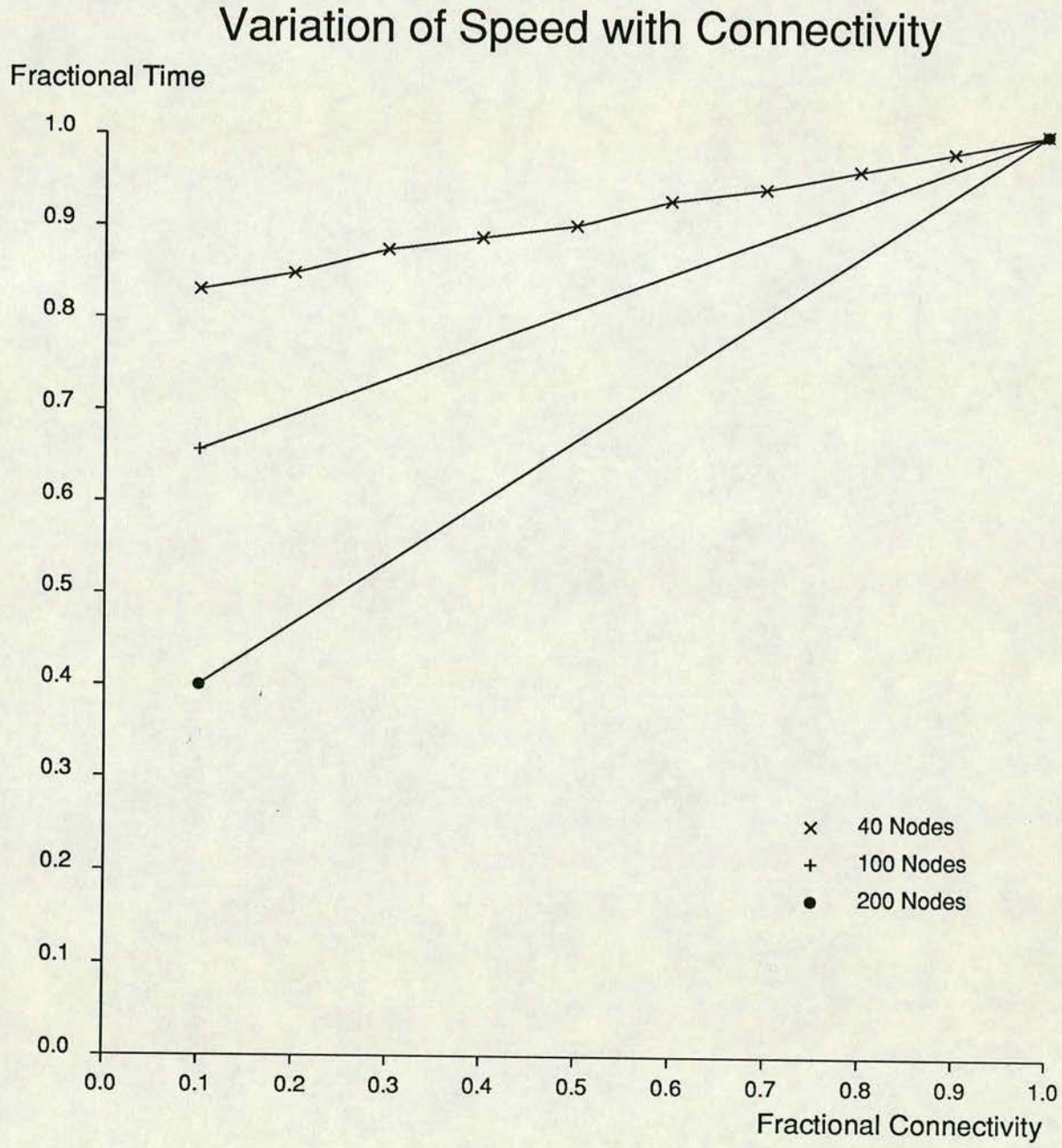

Figure 3.3: The fraction of the time taken for a fully connected network to perform a learning cycle versus the connectivity of the network for various network sizes.

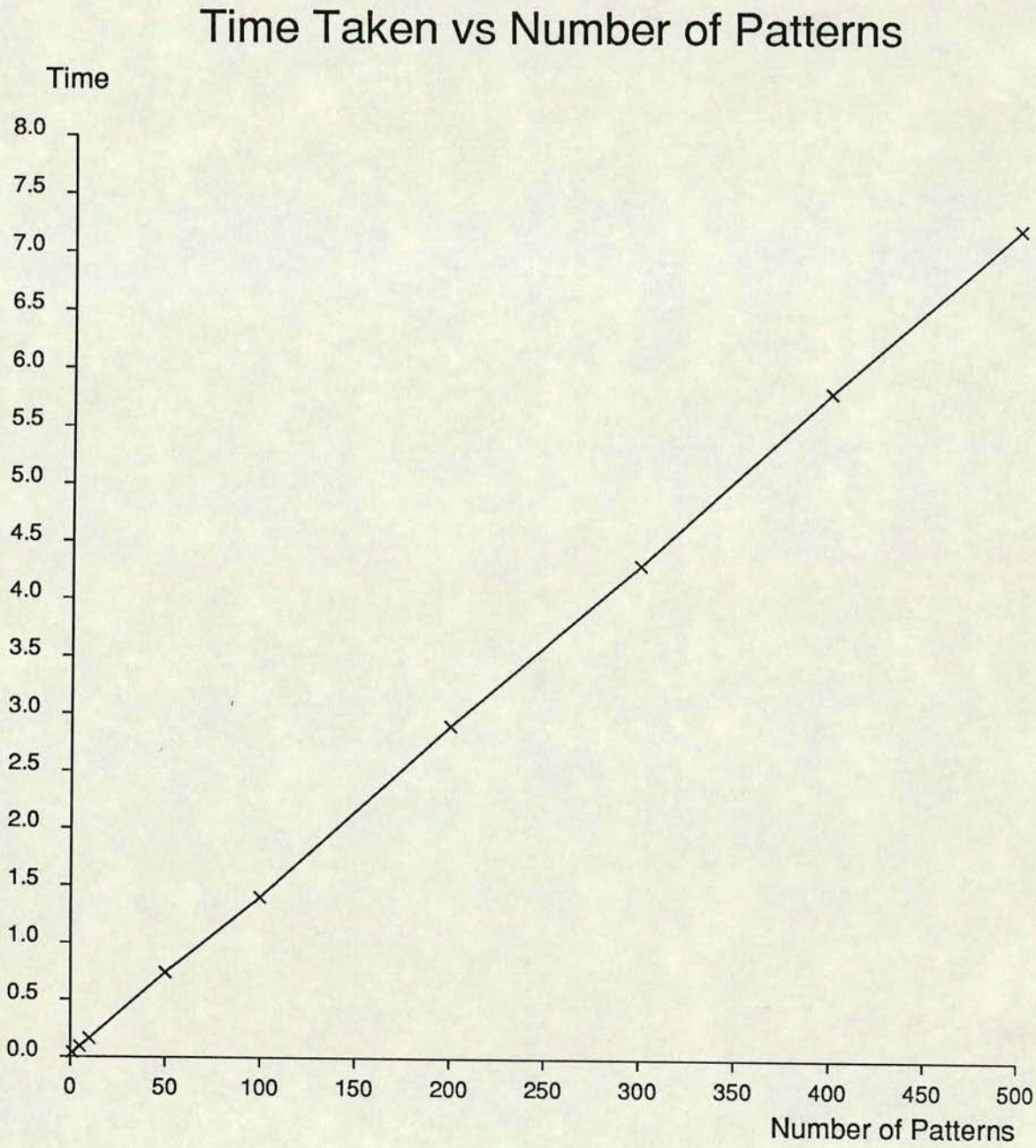

Figure 3.4: The variation of learning time with the number of training patterns. The time (in seconds) is for a 40-40-40 network to do one iteration.

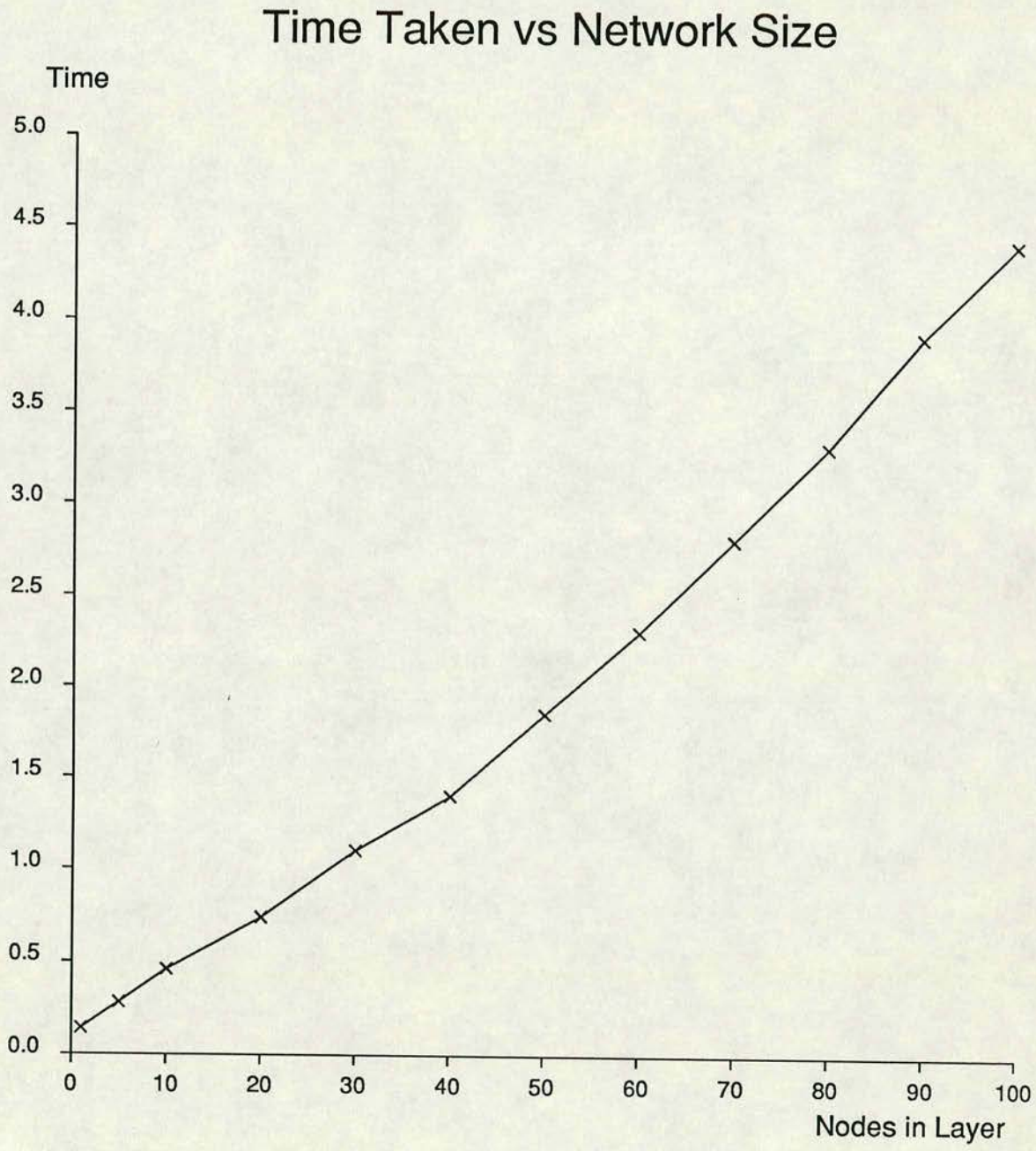

Figure 3.5: The variation of learning time with the number of nodes in each layer. The time is that taken for a one iteration with 100 training patterns.

p	40	100	200
4	302	407	454
9	179	302	384
16	90.4	185	277
25	57.5	128	207
36	32.9	78.3	138
49	23.7	56.8	99.9

Table 3.3: The number of floating point calculations made by the program (given in $kflops$) for various sizes of networks and transputer configurations.

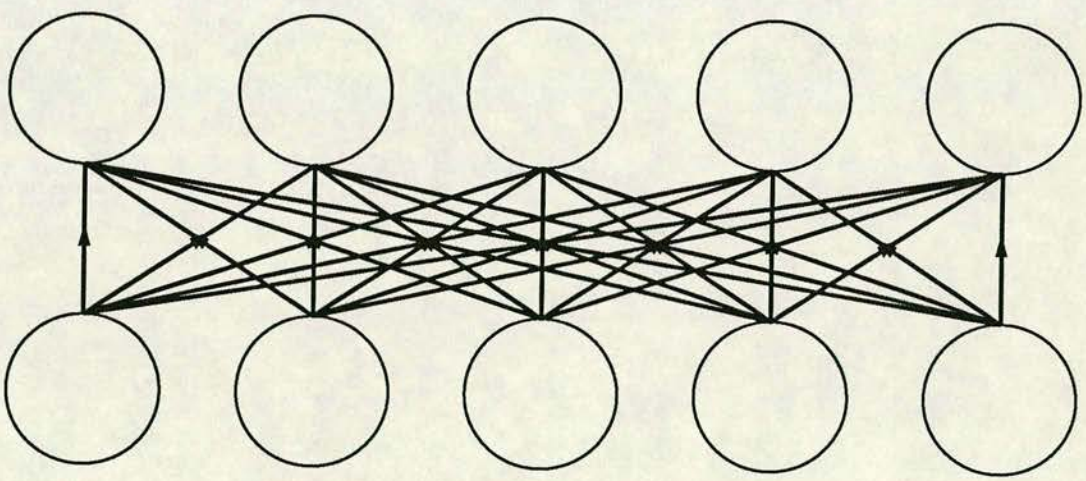

Figure 3.6: A window of connections.

Chapter 4

Generalisation

One of the main hopes with layered networks is that they will be able to generalise. That is, if they are trained on a sample of a large data set, the mapping provided by the network will do well when it is applied to an unseen part of the set. What is meant by doing well? The network is generally judged to have generalised correctly if the generalisation produced is the same as the one desired. A problem though is that the desired function of the network may only be one of many possible functions that would have very similar effects on the training set. The one produced by the network may therefore not be that wanted. In many cases networks are trained on binary patterns which illustrate the problem well . For example if a network is trained on N binary patterns of n bits then these patterns come from a truth table with 2^n entries. If $N < 2^n$ then there are $2^{(2^n - N)}$ possible Boolean functions that could produce the training set. Of these possibilities why should the network produce the one desired? This is illustrated in table 4.1. In this table the training set contains four binary patterns of four bits. These patterns come from a truth table that has eight entries ($2^3 = 8$) and means that there are $2^4 = 16$ possible truth tables or Boolean functions that will produce the training set and so if the network were to choose at random it would have a one in sixteen chance of choosing the one desired. This will grow rapidly worse the larger the training patterns. It is therefore important to choose the training set so that it is representative of the function desired.

In many cases, such as the one described in this chapter, the problem is not as severe.

The generalisation work in this chapter was training a network to classify four images. Each image was of 256^2 pixels of 256 grey levels taken from the Brodatz album (Brodatz 1966). This album contains photographs of natural and synthetic materials which were taken in controlled lighting conditions. The photographs were intended for artists and designers but have been adopted by many people in the image processing world as convenient and standard images. Examples of these images including those used can be seen in figure 4.1. A survey of methods used in the analysis of textures can be seen in Booth(1988) in which the Brodatz textures are also discussed.

The problem was to train the network so that when presented with an n^2 pixel window from one of the images it would classify from which of the main images it came. The method of training was to produce two discrete sets of sample data (that is they contained no common or overlapping images) containing the same number of samples. This was done by dividing the main images in two and choosing at random windows from each half; one formed the training set and the other formed the test set. In these runs both sets contained 512 samples containing 128 from each of the main images.

The networks used were always of the form n^2 input nodes, $n^2/2$ hidden nodes and four output nodes. There was one output node for each of the main images and the task was to train the network to set the node corresponding to the main image when presented with part of that image. Empirically it is found that the number of hidden nodes is not critical. There must not be so few that the data must be compressed too much but it is found that performance is not generally adversely affected if more nodes are added. A discussion of this can be seen in Smieja(1989).

The network was trained on the training data until the error could not be seen to be decreasing; there was no tolerance measure. It was then tested on both the training and test sets. The performance measure used was that the network was taken to have classified the sample as being from which ever group had the most active output node. A matrix was then formed for each test run in which element in row i and column j is the number of samples from image j classified as being from image i. The trace of this matrix if the classification were perfect would

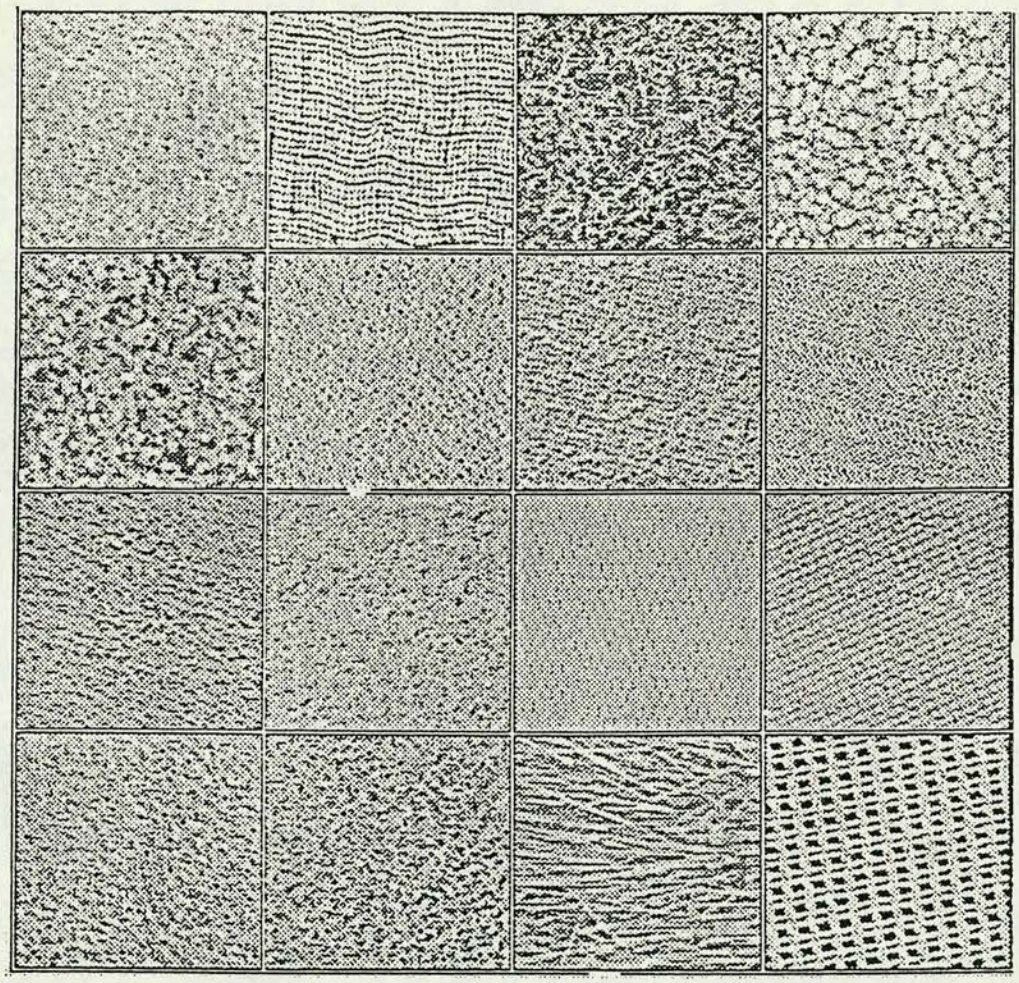

Figure 4.1: Example of Brodatz textures. The top four were used in the runs.

be the number of patterns. A discussion of this and a comparison with human performance can be seen in the work which inspired this texture work. This can be found in Dodd(1988).

Several runs were carried out with values of n ranging from 5 to 20 giving a variation of generalisation performance with window size. Figure 4.2 shows the variation of the trace of the classification matrix with the size of the window. It shows that the trace reaches its maximum value in the case of the training data at about $n = 8$ and remains there thereafter. It also shows that for the test data the performance reaches a maximum at about $n = 11$ of approximately 85% of the patterns correctly classified. If the training data is looked at in more detail by taking the individual textures it can be seen that not all the textures behave in the same way. From figure 4.3 it can be seen that texture one is classified correctly the latest: that is it needs a larger window size than the others, and that all of the textures are correctly classified at a window size of eleven. If the individual test textures are looked at, a similar behaviour can be seen. These are shown in figure 4.4 and it can be seen that maximum values are reached at $n = 11$ and that texture one has the worst performance

4.1 Kurtosis

It might be anticipated that the amount of extra information that is presented to the network rapidly reaches a maximum as the window size is increased. For example if the window size is greater than the maximum correlation length of the image then making it any larger will provide very little extra information and may even make matters worse by making the network too large.

In the images there is no reason to expect the distribution of the pixel values to be Gaussian. If however the images are blocked by a factor of n (that is they are scaled so that one of the new pixels is the average of n^2 of the old pixels) then by the central limit theorem the distribution of these blocked pixels will tend to a Gaussian for large n. The central limit theorem states that if $\{x_i : 1 \leq i \leq N\}$ is a set of N independent variables of mean μ and variance σ^2 then for large n their mean will tend to a normal distribution of mean μ and variance σ^2/n.

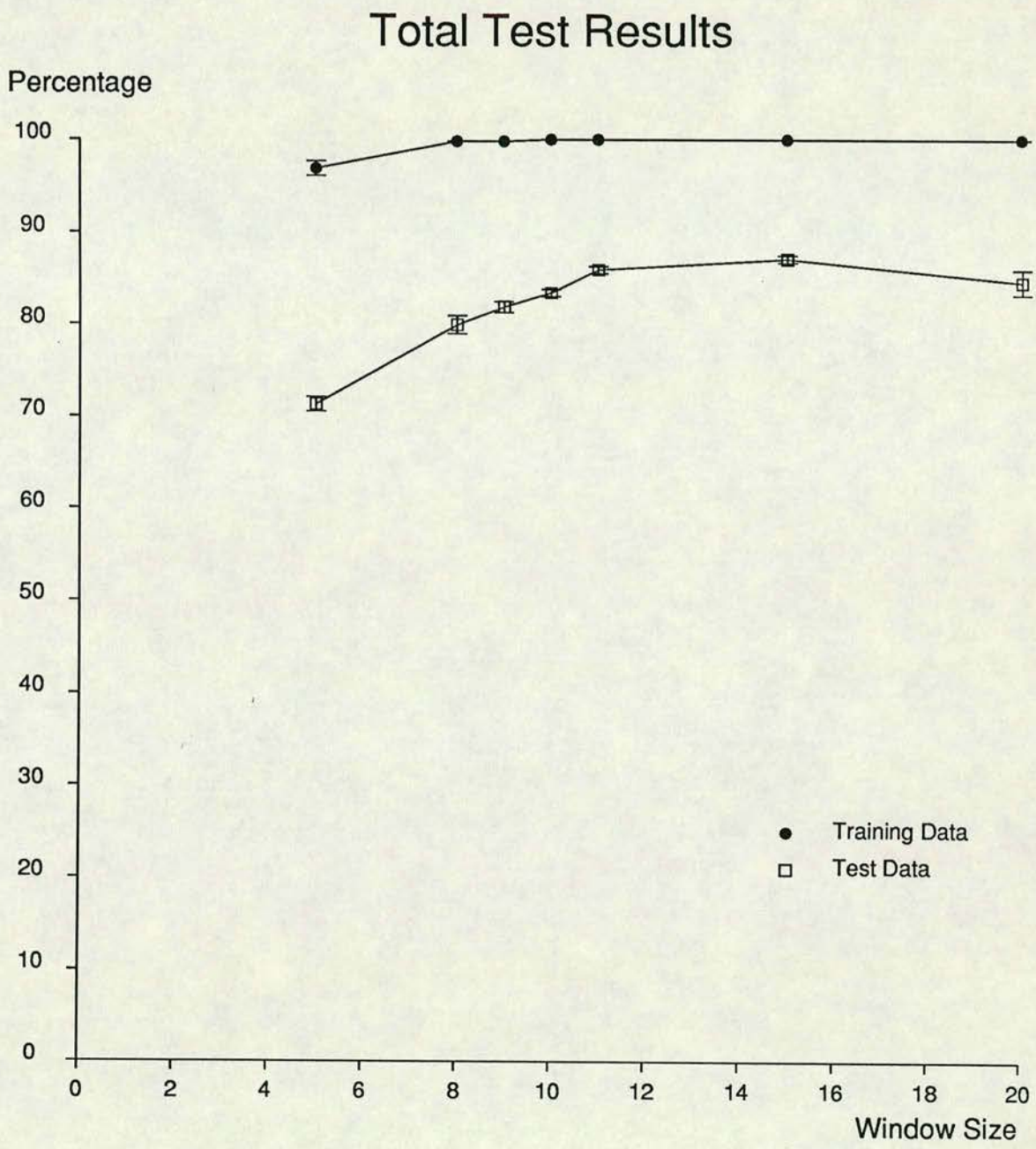

Figure 4.2: The percentage of patterns correctly classified versus the size of the window.

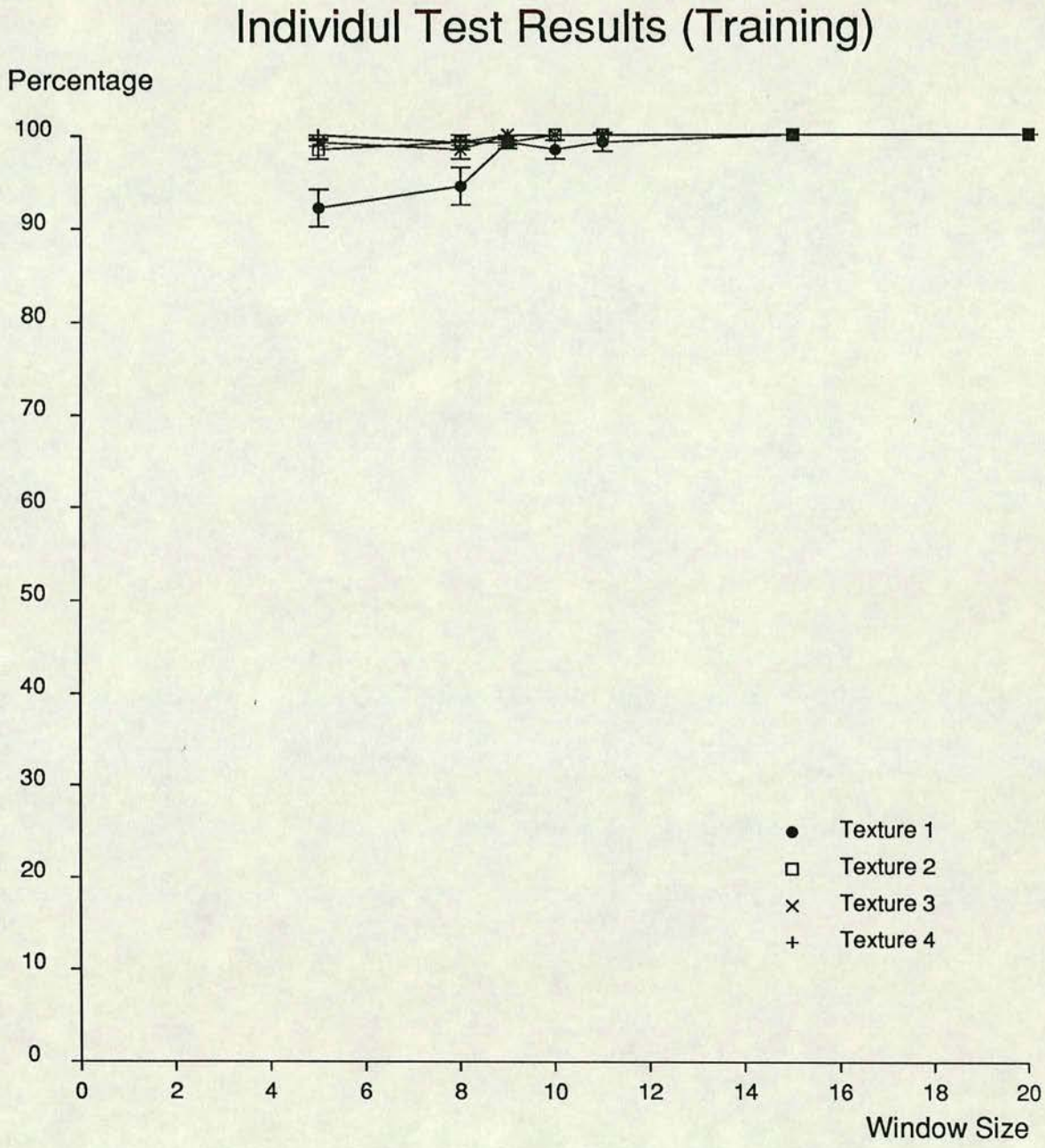

Figure 4.3: The percentage of patterns correctly classified for individual images in the training data versus the size of the window.

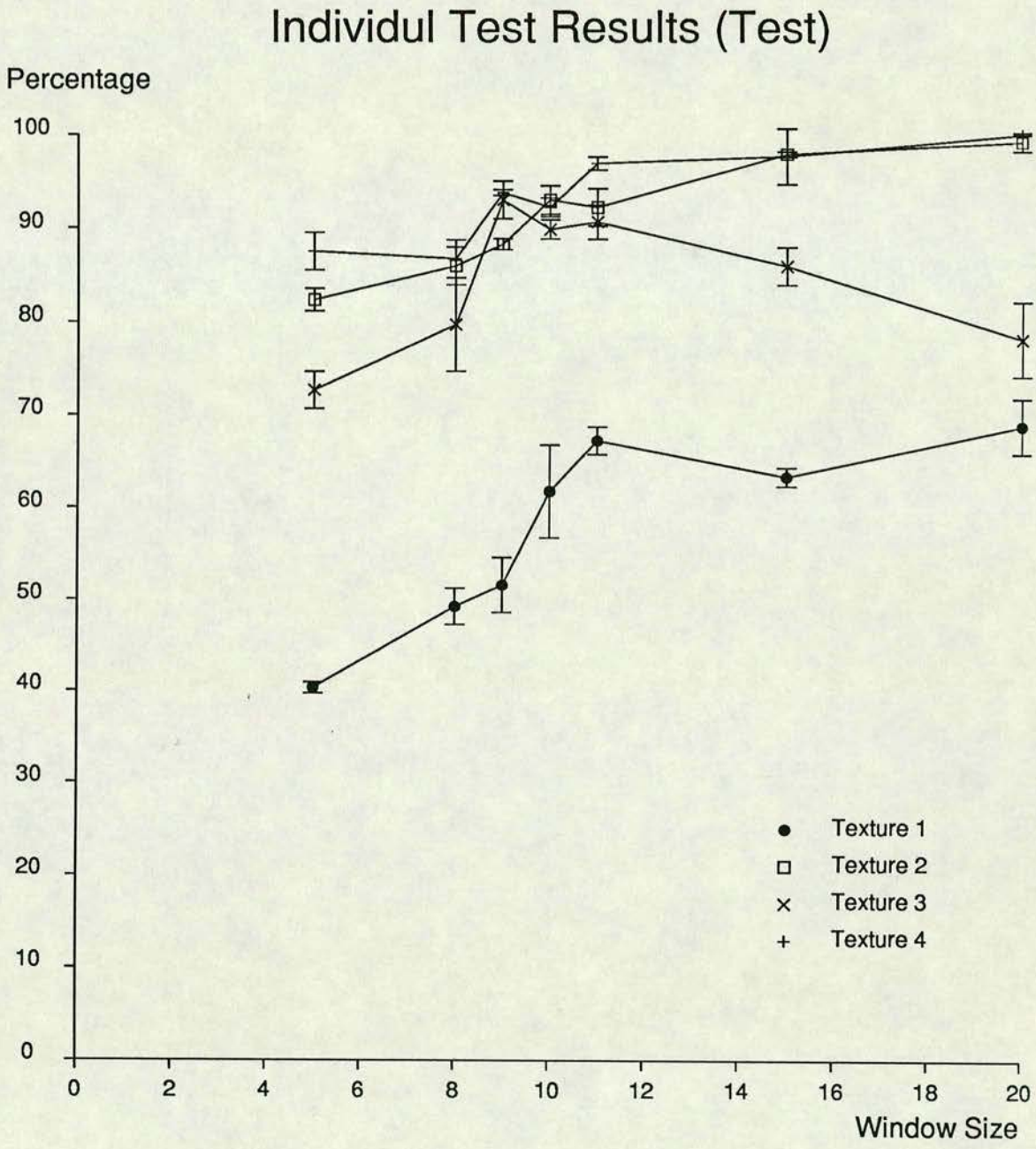

Figure 4.4: The percentage of patterns correctly classified for the individual images in the test data versus the size of the window.

A measure of how near normal a distribution is, is given by the kurtosis of the distribution. Kurtosis is given by the four point cumulant

$$\gamma_2 = \frac{\mu_4}{\mu_2^2} - 3$$

where $\mu_n = E[(x - \mu)^n]$ is the n^{th} central moment. This measures the degree of flattening of the frequency curve near its mode; positive values tend to indicate that the distribution is taller and thinner than a normal distribution and negative values indicate that it is broader. Kurtosis is zero for normal distributions.

Figure 4.5 shows γ_2 for the four textures. From this figure several things can be seen. Texture one has a significantly non-zero kurtosis until a blocking size of about ten. Texture two has a positive kurtosis indicating that its pixel values are concentrated around the mode and is non-zero until about the same point as texture one. The curve for texture three rapidly approaches zero. That for texture four is almost zero from the start.

Some correspondence between figure 4.5 and figures 4.3 and 4.4 can be seen. Maximum performance is reached at a window size of eleven: making the window larger does not improve the generalising abilities of the network. In figure 4.3 it can clearly be seen that the network was not able to learn to recognise the samples from texture one until a window size of about nine to ten and from figure 4.4 it can be seen that the same is true in the testing. From the testing figure it can be seen that as expected the network performed best with texture four. With the other textures the case is not as clear.

4.2 Over Learning

One of the problems encountered in the attempt to make neural networks generalise is that of over learning. This occurs when the network is trained to such an extent that it starts to become so specialised with the training set that its generalising abilities deteriorate. A way of picturing this is if learning is viewed as curve fitting. The entire data set can be thought of as a set of points and the mapping function produced by the network as a curve that is to be fitted through these points. The training set is a subset of these points and at the start of learning

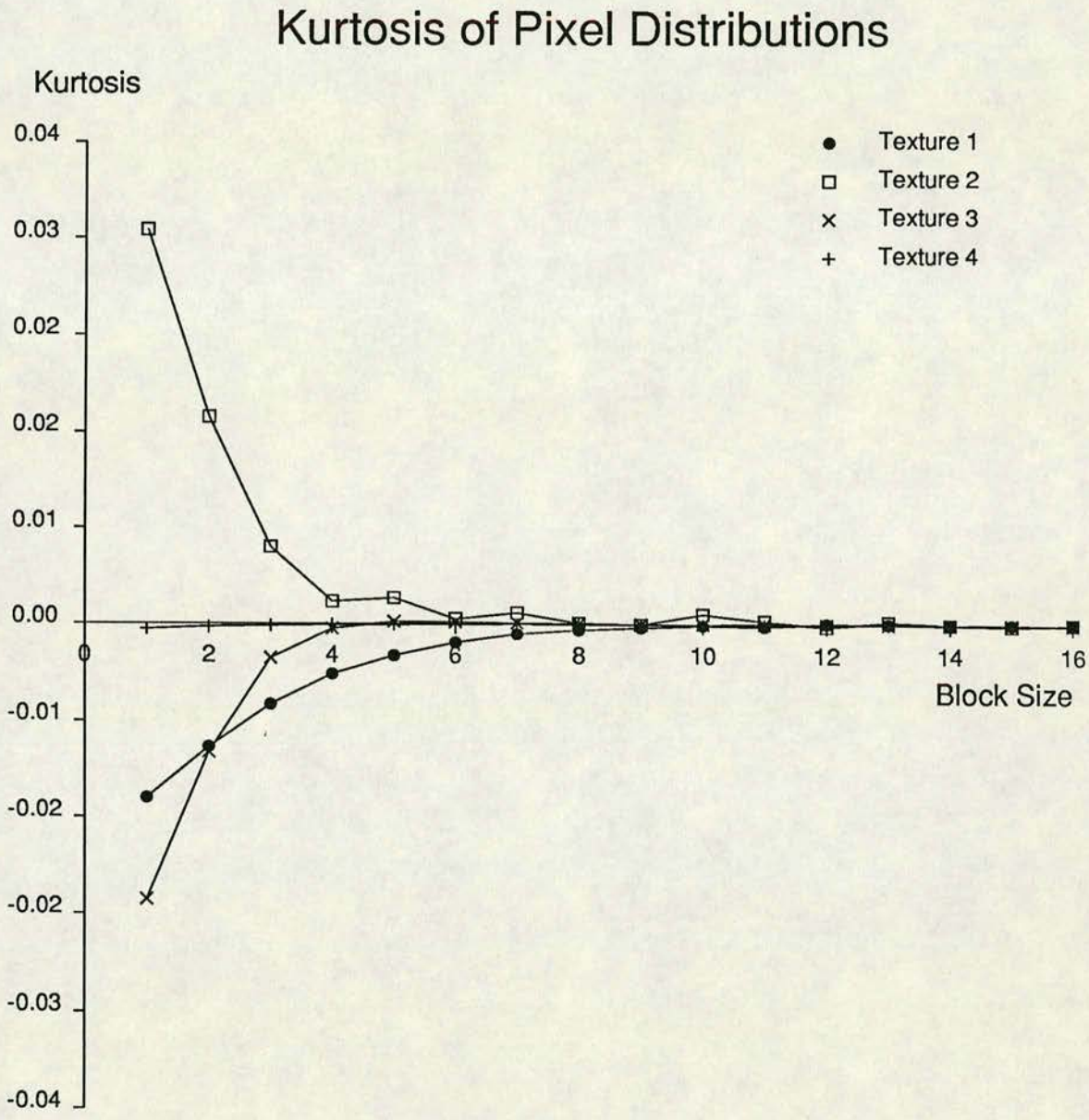

Figure 4.5: The kurtosis of the texture pixel value distributions versus the scale of the blocking.

the function (or curve) will be just as poor for both the training and test set. As learning progresses the curve will become a closer fit and will tend to approach both subsets but if learning continues too far the curve may well become such a good fit to the training set that it will move away from the test set points.

This was encountered in a minor way in this work as can be seen in figure 4.6. This figure shows the variation in the number of samples correctly classified in the latter stages of training. As can be seen, the percentage performance of the network with textures one and three drops from their maxima at an error of about ten. Texture four reaches 100% and stays there and texture three reaches about 98%. Figure 4.7 casts some light on this. It shows the sum of the Euclidean distances of the output vectors from their targets over all the test patterns for each of the four cases. The sum of these is the error the network is minimising in the case of the training patterns. It can be seen from the figure that the total is steadily decreasing and so the total performance is improved at the cost of the performance for individual cases. This is one of the costs of the error function used. Another problem is that the network very easily falls into the local minimum where all the output nodes are set to $1/n$ for an n node output layer. This state has a low error with the cost function used of $\frac{1}{2}\left[1-\frac{1}{n}\right]$. To avoid this minimum it was necessary to start learning with a low value of η (as low as 10^{-5} in some cases) and with a low or zero α. The step size could be increased once it was seen that this minimum was being moved away from. Another common minimum is that where all output nodes are zero. This one does not have quite such a low error and so was not encountered quite so often. The first problem was met by McCulloch *et al.*(1987) but could be avoided by using a different output coding.

4.3 Ising Model Configurations

The above results indicate that correlation lengths in the images can be used to determine how much of the image is needed to train. As a further test of this tests were carried out to test if a network could discriminate samples taken from Ising configurations with correlation lengths of ξ = 2,4,8 and 16. The correlation

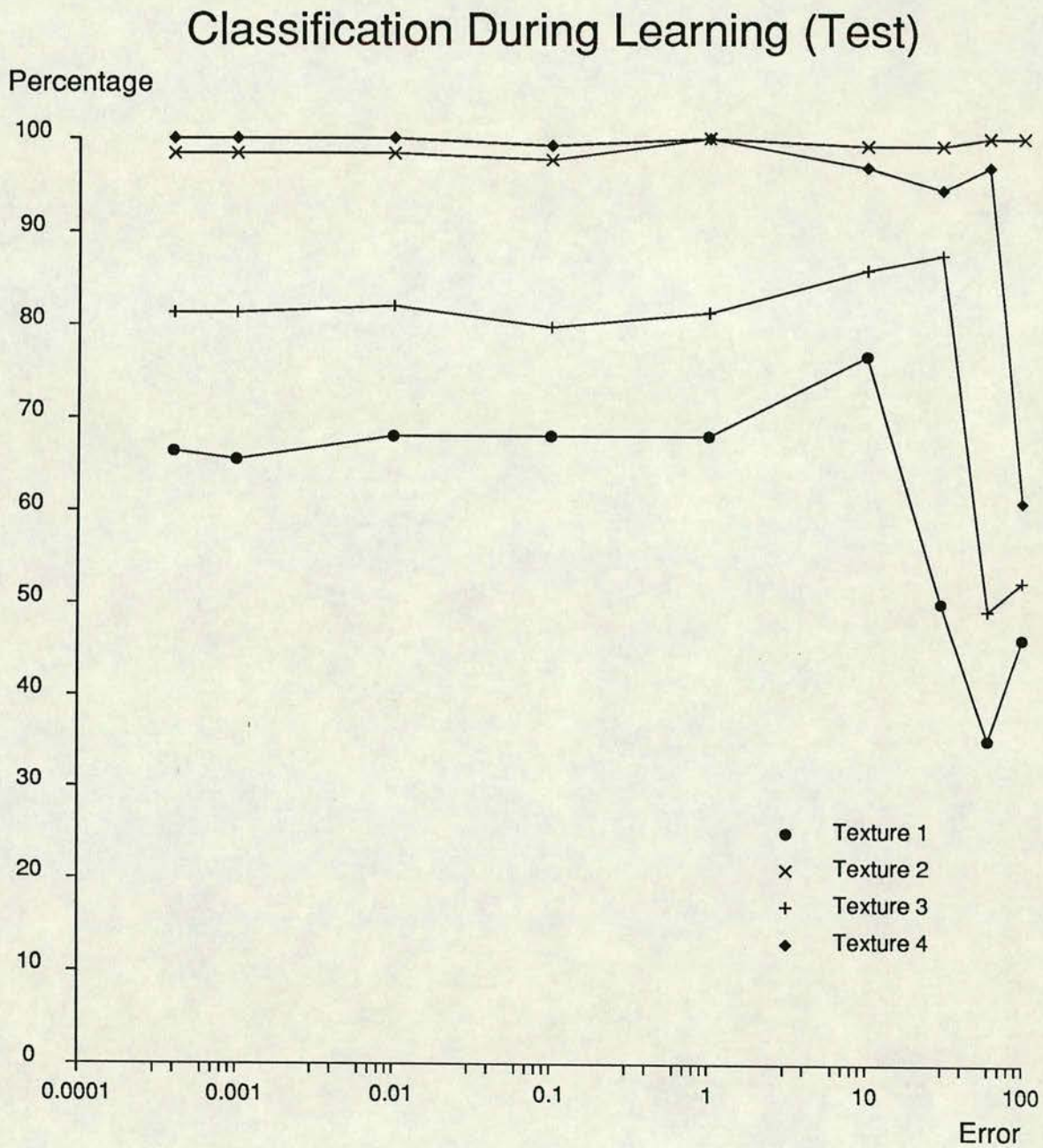

Figure 4.6: The variation of performance for individual test classes during the end of learning. This figure shows how the percentage of the patterns from each of the four classes correctly classified varies towards the end of learning.

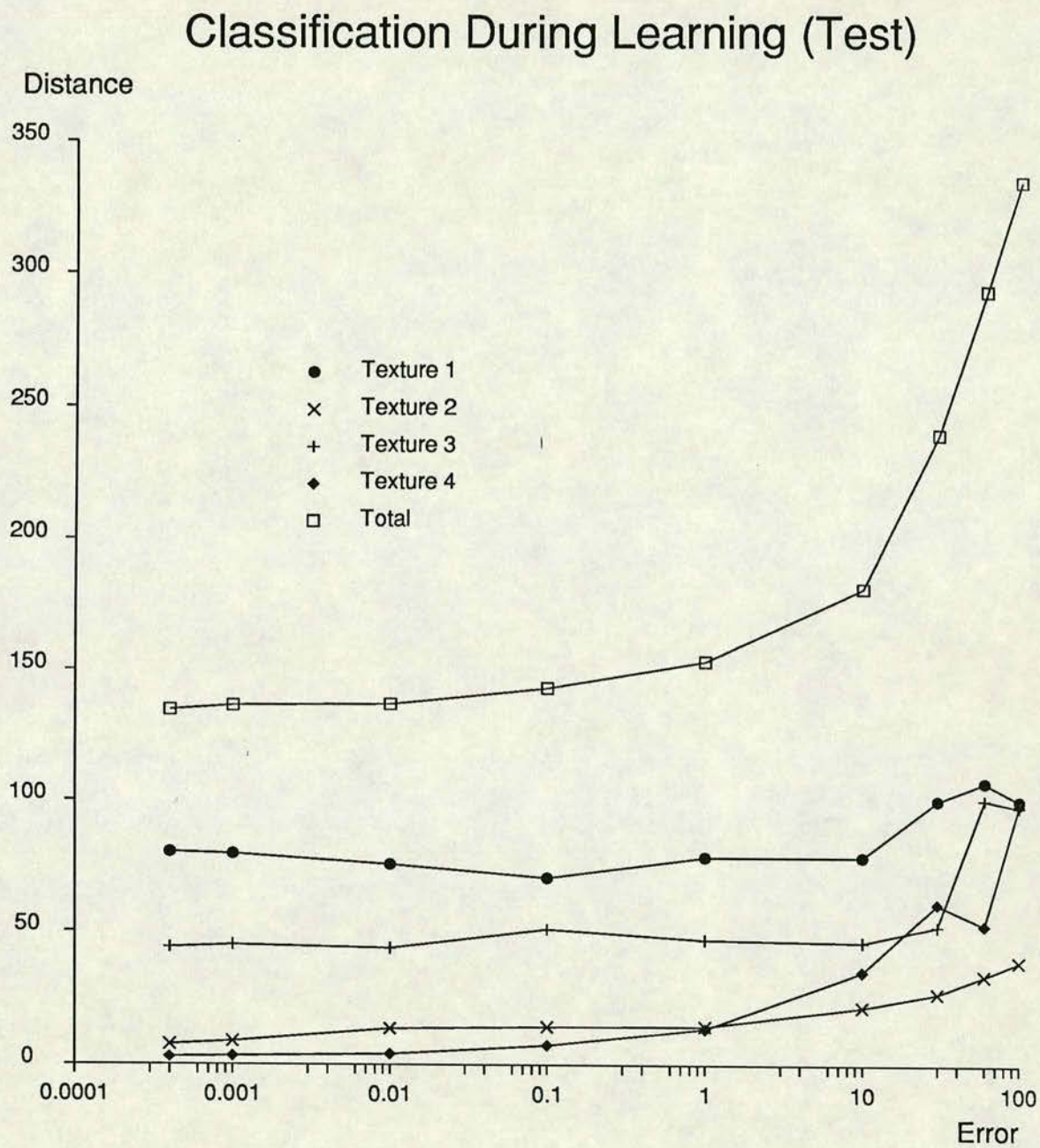

Figure 4.7: The variation of the distance between the output and target vectors for the test set during the end of learning. This figure shows how the Euclidean distance between the actual and target outputs for each of the four classes of texture vary.

length is given by

$$\langle x_i x_j \rangle \sim f(r) e^{\frac{-r}{\xi}}$$

where $\{x\}$ is the set of Ising spins, r is the distance between them and the strongest r dependence is in the exponential. (An investigation of the performance of a neural network in discriminating two one dimensional Ising configurations was carried out by Bedworth(1987).) Runs were first made with 10×10 samples from the four systems with, as above, half the number of hidden nodes. The same training process was used as above with 512 training patterns and the same number of test patterns. The results were very disappointing. After the network was trained it was tested on the training patterns and classified all of them correctly. However on the test patterns it only managed to classify 137 as being from the correct system. Table 4.2 shows the classification matrix for the test set. As can be seen there is no real indication that the network has generalised at all. If the classification had been random then the expected entries would be 32 with a standard deviation of about five. All of the leading diagonal entries are within this.

A method of reducing the degrees of freedom of the network is to reduce the number of weights. It was thought that the network might be forced to generalise if this were done. The network was set up with 10% connectivity and it was found that the performance with the test set was fairly similar with 126 correct classifications. Another way of reducing the number of weights is to reduce the number of hidden nodes. The runs were made with ten hidden nodes but with no improvement.

Another attempt to help the network generalise was made by increasing the size of the training set. The runs were repeated with 2048 patterns, four times as many as for previous runs. In this case 569 patterns were classified correctly which is equivalent to about 142 for 512 patterns: no real improvement.

It may be thought that giving the network the raw Ising data is making the problem too hard by swamping the network with information. As a test of this the runs were repeated but this time with 16×16 windows blocked twice to reduce them to 4×4. This time the network classified 151 of the test patterns correctly. Table 4.3 shows the classification matrix for this run and as it can be seen from comparison with table 4.2 there is some improvement for the $\xi = 2$ and 4 systems

but still no significant generalisation.

It appears as if the task attempted above was too difficult for the network. A neural network does not appear to be able to discriminate to any significant extent Ising configurations of differing ξ's despite the fact that they clearly must have different kurtosis but is it capable of distinguishing a pattern with a given ξ and a random pattern? To test this the network was trained with samples of a pattern with $\xi = 8$ and samples from a hot system (that is random). The samples were of size 10×10 and the training schedule was as above. The network managed to correctly classify all 256 samples from the training set and when tested classified 98 of the random patterns correctly and 103 of the ordered patterns which is far better than would be expected if it were random. It seems as if the network is capable of the task.

4.4 Conclusion

From the above results it appears as if the correlation lengths within the training data do relate to the size of the input vectors needed for successful generalisation. There was found to be no significant improvement in the performance of the network if vectors larger than this length were used. In the case of the texture work fairly good agreement was found between the performance of the network when trained on data of a given window size and the correlation lengths within those textures. No such agreement was found in the case of the Ising data. Even though there was no success in the attempt to make the network discriminate patterns with different ξ's the results with the random system do support the above claims. There was also evidence to support the idea of over-learning. This is the claim that training need not be pushed to the limit since this may result in over learning and a reduction in the generalising abilities.

n_1	n_2	n_3	o
0	0	0	x
0	0	1	x
0	1	0	x
0	1	1	x
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Table 4.1: Here is shown the four patterns in a training set for a network with three input nodes and one output node. These patterns are from an eight element truth table with four undefined entries.

	2	4	8	16
2	37	35	26	25
4	45	31	26	29
8	23	32	35	40
16	23	30	41	34

Table 4.2: Classification matrix for 10×10 window of raw Ising data.

	2	4	8	16
2	47	34	36	23
4	38	32	28	25
8	19	41	23	31
16	24	21	41	49

Table 4.3: Classification matrix for 16×16 window reduced to 4×4 by two blockings.

Chapter 5

Associative Memory

With the conventional type of memory, the type found in computers, a memory location is given and the contents of that location are returned. There is however another type of memory called the associative or content-addressable memory of which there are two main types: the auto-associative and hetero-associative. A typical example of the first kind is the case where a corrupt example of a stored image[15] presented to the storage device and the clean image is returned. In the second case the cue presented and the returned pattern can be completely different. For example, the input pattern might be a face and the output the persons name.

There has been much interest in the capabilities of the Hopfield network as an associative memory and much of this work has been investigating the storage capacity of the network. An important parameter for studying the ability of the network to store patterns is

$$\alpha = \frac{p}{N}$$

where p is the number of patterns and N is the number of nodes in the network or in the general case the number of connections to a node. As α is increased a point is reached where the storage ability of the network begins to break down. The precise value of α at which this happens is dependent upon the dynamics of the network, the network connectivity and the type of synapses permitted. It has been shown (Amit 1985) that for the fully connected Hopfield model (Hopfield

1982, Little 1978) using the Hebbian prescription

$$J_{ij} = \frac{1}{N}\sum_{r=1}^{p} \xi_i^r \xi_j^r \quad (i \neq j)$$

that the network can successfully store up to $p = 0.14N$ patterns. Here the J_{ij}'s are the synapses with $J_{ii} = 0$ and $\{\xi_i^r\}$ are a set of N-bit binary patterns. Due to obvious implementation problems with full connectivity the properties of networks with a limited number of connections per site have been studied (Canning 1988). The properties of networks with discretised connections have also been investigated. For example, in the extreme case of above when the synapses are restricted to ± 1 it has been calculated by Hemmen(1987) that a fully connected network will function as an associative memory storing uncorrelated patterns with a critical storage ratio $\alpha_c = 0.102$. There is of course a limit to the capacity of any network and for these simple networks the maximum value of the storage ratio α was calculated by Gardner(1987, 1988). It was shown that by optimal choice of the synapses the limit of $\alpha = 2$ could be reached.

5.1 Definitions

In the following work two statistical properties of the patterns are used. These are the magnetisation and overlap and are discussed below.

In very simple models of magnetic systems the constituent spins are considered to have two possible orientations, either 'up' or 'down'. The magnetisation is an order parameter which measures the alignment of the spins: the greater the alignment, the greater the overall magnetism. Since there are only two possible values, they can be represented as $+1$ for 'up' and -1 for 'down' and the sum of these divided by the number of spins will then be $+1$ if they are all 'up', 0 if they are random giving no net magnetisation and -1 if they are all 'down'. The magnetisation of a pattern is the mean bit value. Here the bits are taken to be ± 1 whereas in general in layered neural networks they would be 0 or 1. If 0 or 1 is used the node value u_i should be replaced by $2u_i - 1$. The mean bit value or magnetisation of pattern $\mathbf{u}$ is simply given by

$$m_b = \frac{1}{N}\sum_{i=1}^{N} u_i$$

For a number of patterns the magnetisation serves as a measure of their bias or as a correlation measure since if $m_b = 0$ the patterns are completely uncorrelated and if it is ± 1 they are all the same; as correlated as they can be.

The overlap of two patterns is the difference between the number of similar and dissimilar bits in the two patterns. This is the simplest measure of the similarity of the patterns (it doesn't for example take into account the possibility that one pattern is the other shifted). This overlap is one if the patterns are identical, zero if they are uncorrelated and -1 if they are inverses. The mean overlap of pairs of patterns will be a measure of their correlation and for patterns **u** and **v** is given by

$$m_o = \frac{1}{N} \sum_{i=1}^{N} u_i v_i \tag{5.1}$$

It is interesting in the Hopfield model to study the overlap of the spin configuration S with one of the patterns, say ξ_i^1. It has been shown that if the model is iterated then $M(t+1) = f(M(t))$ where $M(t)$ is the overlap at time t

$$M(t) = \frac{1}{N} \sum_{i=1}^{N} \xi_i^1 S_i(t) \tag{5.2}$$

and f is a function that depends only on that overlap. For partially connected networks it has been shown (Gardner 1989) that for $\alpha < 2$ that there is a stable fixed point at $M = 1$ with finite basin of attraction for an initial overlap greater than some minimum M_L. In this regime the system iterates to $M = 1$ and the patterns are recovered. As α approaches two the value of M_L tends to one; the system becomes less capable of correcting flipped spins the more patterns there are stored in the network. If a pattern is entered with overlap less than M_L it iterates to another fixed point at $M = 0$. There is a value of α called α_b below which $M_L = 0$ which means that an input with a finite overlap however small will iterate to the nominal state.

In the fully connected case there are two other interesting quantities: M_0 and M_1. The first of these is the initial value of M above which the pattern will be recovered after complete iteration (this is the equivalent of M_L) and the second is the initial value of M above which the configuration moves towards the pattern in the first time step. Numerical results (Forrest 1988) have shown that M_1 is typically less than M_0 and that for low enough α it can be zero. The fact that

M_1 is less than M_0 can be understood by the fact that if a pattern is presented to the network with random bits flipped then after the first iteration some of the incorrect bits may have been corrected but some of the correct bits may also have been flipped and correlations in the bit values may have been introduced by the non-random weights.

In this chapter the capabilities of the layered feed-forward network when used as an associative memory are investigated. Since a single pass through the network is carried out to produce the output these networks may be expected to behave in a similar way to non-iterative single layer networks. It may be expected that there is a number of patterns below which the output will move towards the target for any non-zero initial overlap by analogy with α_b and the case where $M_1 = 0$. It may also be expected that for a certain number of patterns stored in the network there will be a point equivalent to M_L and that this point would approach one as the number of patterns increases. There may also be a point above which the pattern is completely recovered.

The training set is constructed so that the input patterns have a known mean magnetisation and the pattern-target pairs have a known mean overlap. This allows the correlation and degree of auto-associativity to be controlled. The network is trained until all the patterns are learned to within tolerance or until the error appears to be constant. The network is then tested by adding varying degrees of noise to the training patterns, running them through the network and calculating the mean overlap between the output and target output. From this the information recovered per pattern can be calculated [1] and the value of M_1.

5.2 Notation

In all the following work the mean over all patterns in the training set is used for the magnetisation and overlap and will be denoted by $\langle x \rangle$.

The mean pattern magnetisation is given by $a = \langle m_b \rangle$. This is achieved by setting bits in the patterns with a probability of $\frac{1+a}{2}$ which for an N bit pattern

[1] Note that the output of the network will not be binary and so are rounded before the overlaps are calculated.

then gives a mean bit value of a with a variance of $\frac{1-a^2}{N}$. This can be shown as follows. If b is the number of bits set then

$$P(b=n) = \left(\begin{array}{c} b \\ n \end{array} \right) \left[\frac{1+a}{2}\right]^n \left[\frac{1-a}{2}\right]^{N-n}$$

since n bits are set and $N-n$ bits are unset. This gives

$$\begin{aligned} \overline{b} &= N\left(\frac{1+a}{2}\right) \\ \sigma_b^2 &= N\left(\frac{1+a}{2}\right)\left(\frac{1-a}{2}\right) \\ &= N\left(\frac{1-a^2}{4}\right) \end{aligned}$$

The probability of the magnetisation m_b having a particular value, say x is then given by

$$P(m_b = x) = P\left(b = \frac{N(1+x)}{2}\right)$$

This then gives

$$\overline{m_b} = \frac{2\overline{b}}{N} - 1 = a$$

The variance is given by

$$\begin{aligned} \sigma_{m_b}^2 &= E(m_b^2) - E^2(m_b) \\ &= E\left(\left[\frac{2b}{N} - 1\right]^2\right) - \left(\frac{2E(b)}{N} - 1\right)^2 \\ &= \frac{4}{N^2}\left(E(b^2) - E^2(b)\right) \\ &= \frac{4}{N^2}\sigma_b^2 \\ &= \frac{1-a^2}{N} \end{aligned}$$

The mean pattern-target overlap in the training data is given by $m_p = \langle m_o \rangle$ where $\mathbf{u}$ and $\mathbf{v}$ in equation 5.1 are the pattern and target pair. This parameter governs how auto-associative the memory is. What is meant by this is that if m_p is varied it takes the memory from auto-associative at $m_p = 1$ through varying degrees of pattern-target correlation to hetero-associative at $m_p = 0$ where the patterns and targets are uncorrelated. These pattern-target pairs are constructed by producing the target from the input but with a probability of $\frac{1-m_p}{2}$ of any bit

being flipped. This gives a mean of m_p and a variance of $\frac{1-m_p^2}{N}$ as for a. These can be derived much in the same way as they were for a.

The measure of how clean the input in the patterns are is given by m_i. This is the mean overlap between the patterns presented to the network at testing and the training patterns. These patterns are constructed by flipping bits in the training patterns with a probability of $\frac{1-m_i}{2}$.

The measure of how clean the output is is given by m_f. This is the mean overlap between the output produced by the network from the input and the associated output patterns. If $m_f = 1$ the output mapping is perfect but if $m_f = 0$ it is random.

5.3 Information

It was pointed out by Amit *et al*(1987) that it is better to examine not only the total number of patterns stored by a network but their total information content. This measure takes into account the amount of information stored in the training patterns and the amount of information lost if a pattern is retrieved with errors ($m_f < 1$). In all measures of information, the entropy E of a single stochastic event A which has probability $p = P(A) \neq 0$ plays a fundamental role. It can be thought of as a measure of how unexpected the event is or of the information yielded by it. This function is denoted by $H(p)$ and is defined on $(0, 1]$, that is events with zero probability are not considered, and needs the following properties

non-negative	$H(p) \geq 0$	$\forall p \in (0,1]$
additive	$H(pq) = H(p) + H(q)$	$p, q \in (0,1]$
normalised	$H\left(\frac{1}{2}\right) = 1$	

The normalisation condition is not essential but assigns unit information to events with probability $\frac{1}{2}$: just as likely to happen as not to happen. Wiener(1948) introduced

$$H(p) = -\log_2 p \quad p \epsilon (0,1] \tag{5.3}$$

$$E(A) = -\log_2 P(A) \tag{5.4}$$

which clearly satisfies all three conditions.

Shannon(1948a,b) introduced the concept of the entropy of an experiment. An experiment here is a measurement with several possible outcomes $A_1, A_2, \ldots, A_n$

with probabilities $p_1, p_2, \ldots, p_n$. The Shannon entropy is the sequence of functions

$$H_n : \Delta_n \mapsto R, (n = 1, 2, \ldots)$$

where Δ_n is the set of distributions

$$\Delta_n := \left\{ (p_1, p_2, \ldots, p_n) : 0 < \sum_{k=1}^{n} p_k \leq 1, p_k \geq 0, k = 1, 2, \ldots, n) \right\} (n = 1, 2, \ldots)$$

which need not be complete. The H_n are defined by

$$H_n(p_1, p_2, \ldots, p_n) = \frac{\sum_{k=1}^{n} L(p_k)}{\sum_{k=1}^{n} p_k} \tag{5.5}$$

where $L(x) = -x\log_2 x$ with $0\log_2 0 := 0$.

For complete probability distributions $\sum_{k=1}^{n} p_k = 1$ and so

$$H_n(p_1, p_2, \ldots, p_n) = \sum_{k=1}^{n} L(p_k) \tag{5.6}$$

This can be interpreted as the arithmetic mean of the entropies $(-\log_2 p)$ of the single events with the probabilities as weights. The entropy E of an experiment is, with $p_k = P(A_k)$, given by

$$E(A_1, A_2, \ldots, A_n) = \sum_{k=1}^{n} L[P(A_k)] \tag{5.7}$$

For a pattern with mean magnetisation a there are two possible outcomes from a measurement of a bit: it is either set or not set. As was said above the probability of it being set is $\frac{1+a}{2}$ which gives the probability of it being unset is $\frac{1-a}{2}$. The information per bit per pattern is from 5.7 given by

$$E_{bit} = -\left[\frac{1+a}{2}\log_2\left(\frac{1+a}{2}\right) + \frac{1-a}{2}\log_2\left(\frac{1-a}{2}\right)\right]$$

One of the properties of information is additivity, that is the information expected from two independent experiments is the sum of the information expected from the individual experiments. Here the first experiment is to test whether a bit is set in the input and the second experiment is to test whether the corresponding output bit is the same. So with the definition

$$I(x) := -\left[\frac{1+x}{2}\log_2\left(\frac{1+x}{2}\right) + \frac{1-x}{2}\log_2\left(\frac{1-x}{2}\right)\right]$$

the information per bit in a pattern-target pair without noise is then just

$$s_p = I(a) + I(m_p)$$

and the information in the whole pair is just

$$s = Ns_p$$

This can also be derived by entropy considerations and taking the logarithm of the number of ways of doing the association.

If a noisy pattern is presented at the input then the probability of a bit being set is $\frac{1+am_i}{2}$. The information in a pattern can then be taken to be the information content assuming the pattern is clean less the information needed to recover the pattern from the noise. A discussion of this can be seen in Nadal(1989). This information is given by

$$s_i = I(am_i) - I(m_i)$$

The information lost to noise in the input is then given by

$$s_1 = I(a) - I(am_i) + I(m_i)$$

In the absence of noise the probability of an output node being set is given by $\frac{1+am_p}{2}$ but if the pattern is retrieved with errors ($m_f < 1$) this probability is $\frac{1+am_pm_f}{2}$. The information per bit at the output is then given by

$$s_f = I(am_pm_f) - I(m_f) \tag{5.8}$$

The information lost due to noise at the output is then just the difference between this and the information content of a clean pattern. That is

$$s_2 = I(am_p) - I(am_pm_f) + I(m_f)$$

If a noisy pattern is presented at the input and another noisy pattern is taken from the output then let the information recovered from noise or the reduction in noise be defined as $s_1 - s_2$ or

$$s_r := [I(a) - I(am_i) - I(am_p) + I(am_pm_f)] + I(m_i) - I(m_f) \tag{5.9}$$

This has many interesting properties. If the patterns are unbiased ($a = 0$) then the terms in the square-brackets vanish leaving $s_r = I(m_i) - I(m_f)$ and is independent of m_p. For purely hetero-associative patterns ($m_p = 0$), $s_r = I(a) - I(am_i) + I(m_i) - I(m_f)$ and if noise were presented at the input ($m_i = 0$) then $s_r =$

$I(a) - I(m_f)$. It would be expected that if the network became overloaded that $m_f \to 0$ which would mean $s_r \to I(a) - 1$. For the auto-associative case ($m_p = 1$), $s_r = -I(am_i) + I(am_f) + I(m_i) - I(m_f)$ which is zero if $a = 1$: no information recovered since all the patterns and targets are identical. It is also zero if $m_i = m_f$: same output overlap as input overlap.

5.4 Aims

The network was trained on 8, 16, 32, 64 128 and 256 patterns and then tested on noisy versions of this training data as many times as was necessary to ensure that in all cases there were 256 test patterns. The runs were carried out on a $50 - n - 50$ network[2] with $n = 50, 25, 10$ or 0. It was hoped that some indication of a scaling law for the storage capacity of the network with the number of hidden nodes and m_p could be deduced.

As was said above a measure of the information contained in a pattern is given by s_p. A plot of this can be seen in figure 5.1. Here can clearly be seen some key features. For example the $a = 1$ curve shows that patterns with $m_p = 1$ contain no information as would be expected since all the patterns are the same and the input-target pairs are the same. A point to note however is that as $a \to 1$ the chance of conflict for $m_p < 1$ increases in that the probability of two identical patterns having different targets increases. This sort of conflict can of course not be resolved by the network. The extreme left of this figure ($m_p = 0$) is a pure hetero-associative memory and the extreme right is pure auto-associative and it can plainly be seen that the information per pattern in all cases decreases the more auto-associative the memory becomes. Similarly it can be seen that the information increases the less correlated the patterns become. It would be expected that the network would be able to store more of the patterns that contain less information. So for example it would be expected that the network would be able to store more $a = 0.5, m_p = 0.5$ patterns than $a = 0, m_p = 0$. It may also be anticipated that approximately the same number of patterns requiring the same

[2]The only reason for choosing 50 input nodes is that it is the largest network that could be simulated quickly enough and the approximations used are reasonably accurate.

information could be stored.

5.5 Results

The first runs were done for a network with fifty hidden nodes. Figures 5.2 - 5.4 show the overlap increase for varying initial overlaps of networks trained to store various numbers of patterns. That is, they show the difference between the initial overlap (m_i) and the final overlap (m_f). From figure 5.2 it can be seen that for eight patterns the overlap increases whatever the value of m_i. As the initial overlap decreases from one, the overlap increase increases until m_i is between 0.4 and 0.5, from which point it decreases until m_i is between -0.4 and -0.5. For positive overlap, a positive overlap change means that the input is being moved closer to the nominal state. However, in the case of negative overlap, unless the increase is greater than the magnitude of the initial overlap it is moving the pattern nearer to random noise. In the case of eight patterns this happens just after the -0.1 point. As the number of patterns increases, the maximum overlap increase decreases and the position of this maximum approaches one. For a very large number of patterns the curve tends to a line along the m_i axis. This is as would be expected for this system since there are the same number of hidden nodes as there are input. In this situation a solution to the problem is to produce paths from the input layer through the hidden layer to the output layer as were produced in chapter 2. As more patterns are added to the training set these paths will become stronger and so if a noisy pattern is input the identical pattern will be produced at the output: $m_f = m_i$.

If figure 5.3 is compared with figure 5.2, the situation can be seen to be very different. In this figure, the behaviour in the case where the input and output patterns have an overlap of 0.75 is displayed (Here the bits have a 12.5% chance of being flipped.). Here the eight pattern curve is very similar indicating that the number of patterns is not yet big enough for the change in m_p to have much effect. As the number of patterns increases this effect increases until for 64 patterns the overlap actually decreases. This indicates that some limit of the network is reached between 32 and 64 patterns. For 64 patterns it can be seen that for $m_i > 0.9$ the

overlap increase is positive or near zero; the pattern moves towards a nominal state and for $m_i < 0.9$ it moves away. This is very similar to the behaviour described above for the Hopfield type network where there is a critical overlap M_L for a given α. It can also be seen that the positions of the peaks of maximum overlap change are approaching $m_i = 1$ and its value is decreasing both of which indicate an increasing intolerance to noise which is as would be expected for this 'harder' problem.

As the number of patterns is increased the shape of these curves goes through four main forms. Firstly for very few patterns error correction can take place and so in the positive initial overlap part the value of the overlap change increases from zero at $m_i = 1$ through a maximum. As the number of patterns increases the network becomes less capable of correction (the basins of attraction shrink) and so the value of this maximum decreases and its position moves towards $m_i = 1$. If too much noise is added (m_i is reduced too much) the pattern is now taken away from the nominal pattern; it has fallen into a different basin of attraction. This region grows as more patterns are stored and its minimum, the point of greatest overlap decrease, moves towards $m_i = 1$. Before it reaches this point however the capacity of the network is reached and the $m_i = 1$ value of the overlap change starts to drop and the shape of the curve approaches the $m_f = 0$ line.

In the negative region there are corresponding changes. For a small number of patterns there is a minimum which broadens as the number of patterns increases and the curve tends to the line $m_f = 0$ for very large numbers of patterns. This line corresponds to the output of the network being uncorrelated with the input. It would be expected that as more patterns are stored in the network that it would become increasingly likely that if a noisy pattern is entered it would fall into the wrong basin of attraction and since the trained patterns are uncorrelated the output would as a result be uncorrelated with the nominal state.

The effects described above become more pronounced for fewer patterns as the value of m_p decreases. This is understandable due to the increased difficulty of the task. This can be seen from figure 5.4 which is for the purely hetero-associative case.

The region of figure 5.2 near $m_i = 1$ can be seen expanded in figure 5.5. It can

be seen that the point at which m_f becomes less than one approaches one as the number of patterns increases. This means that as the network is made to store more patterns the degree of noise at which the pattern will be recovered becomes zero. This is similar to the effect described for the Hopfield network.

It is interesting to compare the overlap change with the information recovered from the noise (equation 5.9) for various values m_i. Figures 5.6 - 5.10 show the the value of s_r for the mean overlap values per pattern. From figure 5.6 it can clearly be seen that the point of maximum information gain approaches $m_i = 1.0$ as the number of patterns stored in the network increases and that this point does not coincide with the point of maximum overlap increase. The maximum of information gain occurs at a larger value of m_i and drops much more rapidly. This means that the network performs with greatest efficiency at a point before the maximum overlap increase. For values of $m_p < 1$ it can be seen that for $m_i < 0$ the information recovered is negative, [3] so that even though the overlap is increased, information is lost due to the fact that the pattern is moved towards an uncorrelated pattern. An interesting aspect of these figures can be found in the negative m_i region. It can be seen that the behaviour is not monotonic in the number of patterns. In the case of figure 5.6 this can be explained as follows. As was said earlier, in this case a solution is to form non-intersecting paths from an input node to the corresponding output node and that these paths will become reinforced the greater the number of patterns. So in the limit of a large number of patterns, the same output will be produced as input and so no information will be recovered or lost. For small numbers of patterns, these paths will not be well developed and so in the negative m_i region the output is very likely to be moved away from the nominal state resulting in information loss. This effect will become less pronounced as m_p decreases resulting in the behaviour becoming more monotonic in the negative region. In the overlap change figures if random noise is presented to the network, the overlap change is positive. If the corresponding information gain and overlap change figures are compared it can be seen that despite this fact, the information gain is zero.

As was stated earlier, for the auto-associative case ($m_p = 0$) and unbiased

[3]This point is actually slightly less than zero but tends to zero for large numbers of patterns.

patterns, a network would be expected to behave such that $s_r = I(m_i) - I(m_f)$. In the case of unbiassed patterns, $I(m_i)$ is equivalent to the information lost due to noise at the input. As was also stated, for large numbers of patterns this will tend to $s_r = I(m_i) - 1$ since $m_f \rightarrow 0$. This behaviour can clearly be seen in figure 5.11.

If it is desired to reduce the noise in the pattern then it can clearly be seen that there is a maximum number of patterns that can be stored in the network. This can be seen clearly from figures 5.12 - 5.16 which show how much information is recovered from patterns with various levels of noise against the number of patterns stored. These figures can be thought of as vertical slices through figures 5.6 - 5.10. An interesting aspect is their non-monotonic behaviour with respect to m_i. This can be explained by the fact that for large m_i there is relatively little information to be recovered. From figure 5.12 it can be seen that in the purely auto-associative case the information recovered approaches zero as the number of patterns increases. This is what would be expected from previous arguments. For non auto-associative memories ($m_p < 1$) the information recovered goes negative and the point at which this occurs decreases with m_i and appears to approach a minimum. The point of this crossing can be used as a storage capacity measure and can be estimated by doing polynomial fits and then calculating the roots of the polynomials. [4] Figure 5.17 shows the values of the roots in the relevant area against the initial overlap wanted (the degree of noise). From this figure it can clearly be seen that the storage capacity of the network drops rapidly for $m_i \approx 1$ and for $m_i < 0.7$ the capacity is fairly flat. It can also be seen that there is ordering with respect to m_p which indicates that the more hetero-associative the memory becomes the less tolerant it is to noise. As has been said, in the Hopfield network, there is a number of patterns given by α_b below which a pattern will move towards its nominal state. There is evidence here of analogous behaviour in that if fewer patterns than the number giving these intercepts are stored then information is recovered.

Table 5.1 shows for fifty hidden nodes how the mean final overlap for 256 patterns varied with the pattern-target overlap. From this table it can be seen

[4]This was done using the NAG routines e02acf and c02adf.

that the overlap as expected tends to decrease as m_p decreases. If however the information stored per pattern per bit is looked at it can be seen that the network is in fact storing more information. This information in the pattern-target pair (s_p) less the information lost due to the noise (s_2) at the output.

$$s_t = s_p - s_2$$

This differs from s_r which is the information recovered from a noisy pattern whereas s_t is the information stored per pattern. That is s_2 deals with clean patterns.

m_p	m_f	s_t
1.0	1.000	1.000
0.75	0.975	1.447
0.5	0.948	1.637
0.25	0.937	1.753
0.0	0.938	1.800

Table 5.1: How the mean final overlap of the output and target output varies with the patter-target overlap for fifty hidden nodes. Also shown is the variation of the information stored in the network per pattern and bit.

The same runs were also done for 25 hidden nodes. Figures 5.18 - 5.19 show the information recovered from noise in this case. If figures 5.18 and 5.6 are compared it can be seen that the performance of the networks for very few patterns is almost identical: the effect of the fewer hidden nodes has not yet become apparent. This effect does become apparent the greater the number of patterns and the information recovered goes negative. In this case, unlike fifty hidden nodes, input-output paths are not possible and so it would be expected that the behaviour would be very different for large numbers of patterns. The effect is however not very great as can be seen in figure 5.20. If this is compared with figure 5.17 it can be seen that the extra nodes give little benefit: hardly any more patterns can be stored. From these figures it can be seen that for fairly clean patterns the performance of the fifty hidden node network is significantly better than that of the smaller. Also if figure 5.10 is compared with figure 5.19 it can be seen that the latter has very bad performance for large numbers of patterns.

Table 5.2 shows the same information as table 5.1 but for 25 hidden nodes. As can be seen less information is stored than for 50 but as for 50 the information stored increases as m_p decreases.

Runs for purely hetero and auto-associative memories were carried out for networks with ten hidden nodes and no hidden nodes. In the case of ten hidden nodes, for auto-associative memory, the severity of the bottle neck is not felt for eight patterns as can be seen by comparing figures 5.6 and 5.21 in which the curves are almost identical. However in the case of sixteen patterns there are too few hidden nodes and since the data cannot be compressed they cannot quite be stored. In the case of the hetero-associative memory the performance is, as would be expected, even worse. It is interesting to compare the no hidden node and fifty hidden node cases (figures 5.6 and 5.23). It can be seen that the forms of the curves are very similar but in the no hidden node case they are not as peaked and the peaks are slightly closer to $m_i = 1$. A possible way of interpreting this is to picture the two layer network as a one iteration network and the three layer network as one with two iterations. In this analogy it may be expected that if the pattern is being moved towards a nominal state it would be moved closer in two iterations; hence the higher peaks. It was said above that it is found in the Hopfield network that the overlap above which patterns are moved towards the nominal states in one iteration (M_1) is lower than the overlap above which patterns move towards the nominal states after complete iteration (M_0). If the analogy above is accurate then it would be expected that the peak would have shifted the other way. The same behaviour is seen in the hetero-associative case (figure 5.24).

If is of interest to study the behaviour of the network when storing correlated or biased patterns ($a > 0$). The network was trained with fifty hidden nodes on auto-associative data but with $a = 0.5$: bits have a 75% chance of being set. Figure 5.25 shows the test results for such a network and as can be seen the heights of the peaks, especially for small numbers of patterns, are significantly reduced. This means that the network if less capable of reducing the noise in the input. From equation 5.9 it can be seen that in the case of $m_p = 1$, $s_r = I(m_i) - I(am_i) - I(m_f) + I(am_f)$. If random noise is entered then this reduces to

$I(am_f) - I(m_f)$. For a large number of patterns it would be expected that in this case $m_f \rightarrow 0$ and so $s_r \rightarrow 0$. It can clearly be seen from figure 5.25 that this is the case. Also if the inverses are entered ($m_i = -1$) then $s_r = I(am_f) - I(m_f) - I(a)$ and if $m_f \rightarrow 0$ as the number of patterns increases then $s_r \rightarrow -I(a) \approx 0.811$ and this is as can be seen from the figure. This shows that unlike in the Hopfield model using the Hebb rule, the inverses of patterns are not stored. This can be easily understood by the lack of symmetry in equation 1.13: making all the node values their inverse does not make $\phi \rightarrow -\phi$.

Figure 5.26 shows the performance of a network when trained on biased patterns with $m_p = 0.5$: not auto-associative. If this figure is compared to the corresponding figure for unbiased patterns (figure 5.8) dramatic differences can be seen. Firstly as above it can be seen that the error-correcting abilities of the network are greatly reduced. It is of interest to compare figures 5.26 and 5.27 in the positive m_i region. In figure 5.27 it can be seen that the overlap is increased for eight patterns for all m_i but from figure 5.26 it can be seen that the information gain is negative for m_i less than about 0.3. The increased overlap is not enough to compensate for the $I(am_i)$ term in equation 5.9. In biased patterns the overlap can be increased by setting bits in the direction of the bias but this does not recover lost information. In a similar way as above, it can be shown that for $m_i = 0$, s_r should tend to about -0.14 and for $m_i = -1$ to about -0.954 for large numbers of patterns which appears to be the case.

Figure 5.28 shows the performance of the network with even more biased patterns ($a = 0.75$). If this figure is compared with figure 5.26 it can clearly been seen that the point at which the information gain becomes negative is moving towards the $m_i = 1$ point or in other words the network is becoming increasingly less tolerant to noise the more biased the patterns become. As was said earlier, in the non-auto-associative case, there may arise the problem of conflict if the patterns are biased: the same input pattern may appear with more than one output pattern. This is probably the cause of the reduced storage capacity in figure 5.28 where the bits have a probability of 88% of being set.

Forrest(1989) used a measure of the performance of an associative memory called R_2. This was the total information stored in the network per bit used to

store it. This could be measured since the weights used were binary. Here however the weights are (at least in principle) continuous. A similar measure can be used: the total information stored per weight. This is just given by

$$s_T = \frac{ps_t}{N_t}$$

where p is the number of patterns stored, m_f is the mean final overlap in the case of the stored patterns and N_t is the total number of weights and biases. Table 5.3 shows the value of m_f for the trained hetero-associative memories. From this table it can be seen that the overlap decreases as the number of patterns increases and as the number of hidden nodes decreases except that the network with no hidden nodes does better than that with ten since it is not forced to compress the patterns. If this table is compared with table 5.4 which shows the information stored per weight it can be seen that the network with ten hidden nodes performs best and the no hidden node and twenty-five hidden nodes perform about the same. The fifty hidden node network does worst of all. This shows that the networks with fewer hidden nodes use the weights more efficiently. That is, if more hidden nodes are used, the output is cleaner at the cost of the efficiency of the weight use. Table 5.5 shows the same information as table 5.4 but for the auto-associative case. In this case the desired input and output are the same. The behaviour similar to the hetero-associative case except that for large numbers of patterns the ten hidden node network is out-performed by the twenty-five and ten node networks. This could be explained by the fact that this network is loosing the most information by compressing the patterns through the hidden layer.

Figures 5.29 and 5.30 show the number of patterns that can be stored in the network for some error recovery for the auto and hetero-associative cases for the different numbers of hidden nodes. That is, it shows the number of patterns that can be stored if the overlap at the output is not going to be less than that at the input. From figure 5.29 it can be seen that the two curves (twenty-five and ten hidden node) behave very differently. These curves were produced in the same way as figures 5.17 and 5.20, that is by polynomial fit. The twenty-five node curve decreases from a maximum at $m_i = 1$ whereas the ten hidden node curve has its minimum there. The reason for this difference in behaviour is not clear. They both end up at about the same point for $m_i = 0$ of about thirty-five patterns.

As was said above, in the Hopfield network there is a value of α called α_b where a pattern with an overlap, however small, will iterate towards its nominal state. Here there is evidence of a similar behaviour. There appears to be a number of patterns (around thirty-five) below which a pattern will move towards its nominal state. This is similar to the behaviour of the fully connected Hopfield network where M_1 can be zero for low enough α. Figure 5.30 shows the scaling with hidden nodes for the hetero-associative network. From this figure and table 5.4 it can be seen that for for small numbers of patterns the order of the information stored per weight corresponds to the order of the number of patterns that can be stored for some error recovery for small amounts of noise. This indicates that the systems in which the weights are being loaded the most are least capable of error correction. This correspondence can also be seen between figure 5.29 and table 5.5.

5.6 Conclusion

In this chapter it has been shown that if the information content of patterns to be stored is considered then quite a different light is thrown upon the performance of the network. If just the mean final overlap is taken into account then it appears in the hetero-associative case as if the more hidden nodes there are then the better the performance. In this situation however having fewer nodes can mean that those nodes are being put to better use. It was also shown that the point at which the network works most efficiently in information terms is not the point at which the greatest overlap increase occurs. With the training data, as the problem is made less auto-associative, the mean final overlap decreases but the information stored per weight increases. Considering the information can also account for the improved performance of the network and the increased storage capacity. It has also been shown that the less auto-associative the problem, that fewer patterns can be stored for error reduction to occur. This indicates that the more information contained in a pattern, the fewer of those patterns can be stored.

For networks with the same number of hidden nodes as input the solution to the auto-associative problem is trivial: non-intersecting paths can be produced from

input to output. These paths cause whatever is put into the input is produced at the output resulting in the output overlap being the same as the input. As the problem is made less auto-associative, these paths become less effective as a solution and the network can become overloaded if too many patterns are stored. When this happens the output overlap becomes less than the input meaning the output is more corrupt than the input.

The changing behaviour of the network with respect to the number of hidden nodes is very interesting. It was shown that as the number of hidden nodes is reduced, the network does perform quite as well but the weights are used more heavily. If the number of patterns that can be stored for noise reduction to take place is plotted against the initial overlap there appears to be a dependence of the gradient on the number of hidden nodes. The results of this work are not sufficiently clear to ascertain what this dependence is. A possible correspondence between the information stored per weight and the number of patterns that can be stored for error recovery to take place was also observed. The addition of more hidden nodes appears to only be of significant benefit for fairly clean patterns. The difference in performance between the fifty and twenty-five hidden node networks for very noisy patterns was not great. It was also shown that for very few patterns networks with more hidden nodes did not perform significantly better.

In the case of biased patterns it has been shown that the performance of the network rapidly deteriorates as the bias grows stronger. It has been shown that information considerations in this case show that the network is not functioning anywhere near as well as the overlap changes make it at first appear.

Many analogies have be drawn between the performance of layered networks and the single layered Hopfield type of network. Direct comparisons are difficult to make since absolute storage capacities were not derived. It has also not been possible to test how the number of patterns storable by the networks depends upon the information content of the pattern-target pair due to the absence of absolute capacities. As has been said, in the Hopfield model for a number of patterns below some critical value there is an initial overlap M_L above which a pattern iterates towards its nominal state and below which it iterates towards zero overlap. As the number of patterns stored in the network increases, the value of this overlap

tends to one. It has been shown in this work that in the type of network studied here there are points of maximum overlap increase and of information recovered and as the number of patterns stored in the network is increased that this peak moves towards an initial overlap of one. It has also been shown that as the number of patterns increases the height of this maximum decreases until above a certain number of patterns the curve drops below the axis. Also in the Hopfield network there is a value of α below which a initial pattern with a finite overlap, however small, will iterate towards its nominal state. It was shown for these networks that there exists a number of patterns below which information will be recovered from an input with a finite overlap. It was said that it may be possible to picture the networks studied here as two iteration networks if there is a hidden layer and one iteration networks if there is not. In the fully connected Hopfield model, the two overlap values M_0 and M_1 have different values. It was not possible to effectively investigate the existence of similar behaviour but there was an indication of a possible shift of the maximum information gain peak. This shift was however very slight and so could not be taken as definite.

It was shown that at least in one respect the network studied differed from the Hopfield network in that pattern inverses are not stored.

The measure used in this chapter of the association between input and output could only be used with networks that have the same number of input and output nodes. It would be interesting to study networks where this is not the case. The network has only been trained on patterns without noise and it would be of interest to investigate the effect of noise on performance.

m_p	m_f	s_t
1.0	0.950	0.831
0.75	0.845	1.150
0.5	0.806	1.352
0.25	0.807	1.496
0.0	0.679	1.365

Table 5.2: How the mean final overlap and information stored vary for twenty-five hidden nodes.

$n\backslash p$	8	16	32	64	128	256
0	1.000	1.000	1.000	0.979	0.877	0.427
10	1.000	0.988	0.956	0.792	0.587	0.047
25	1.000	1.000	0.988	0.976	0.923	0.679
50	1.000	1.000	1.000	0.992	0.971	0.938

Table 5.3: The variation of the mean final overlap after training on various numbers of hetero-associative patterns for different network sizes.

$n\backslash p$	8	16	32	64	128	256
0	0.006	0.013	0.025	0.048	0.084	0.114
10	0.015	0.029	0.056	0.092	0.153	0.242
25	0.006	0.012	0.024	0.047	0.088	0.136
50	0.003	0.006	0.013	0.025	0.047	0.090

Table 5.4: The variation of the total information stored per weight after training on various numbers of hetero-associative patterns for different network sizes.

$n\backslash p$	8	16	32	64	128	256
0	0.003	0.006	0.013	0.024	0.048	0.091
10	0.008	0.015	0.027	0.034	0.054	0.051
25	0.003	0.006	0.012	0.025	0.047	0.083
50	0.002	0.003	0.006	0.013	0.025	0.047

Table 5.5: The variation of the total information stored per weight after training on various numbers of auto-associative patterns for different network sizes.

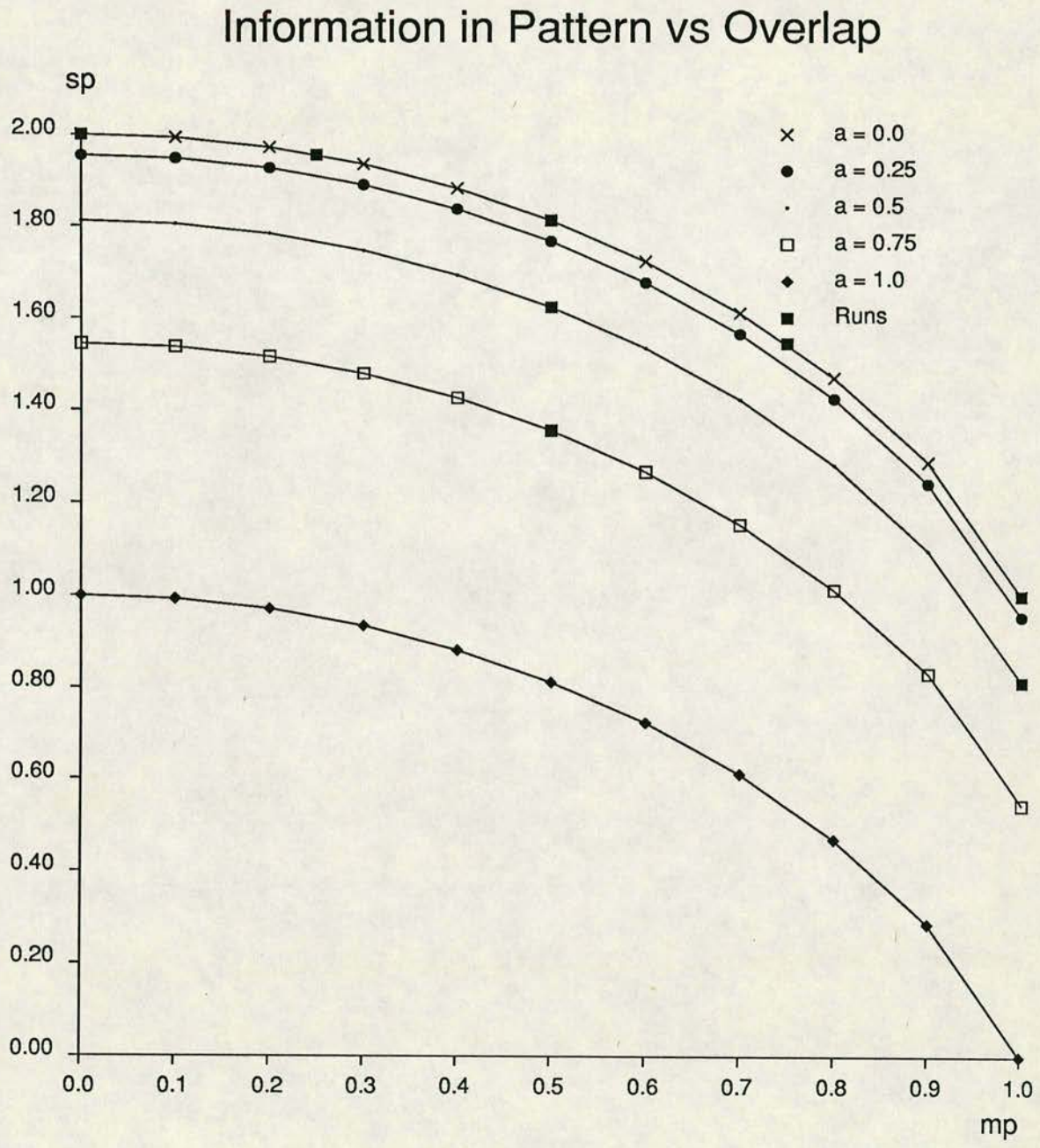

Figure 5.1: Information required to store a pattern for various pattern magnetisation and pattern-target overlaps

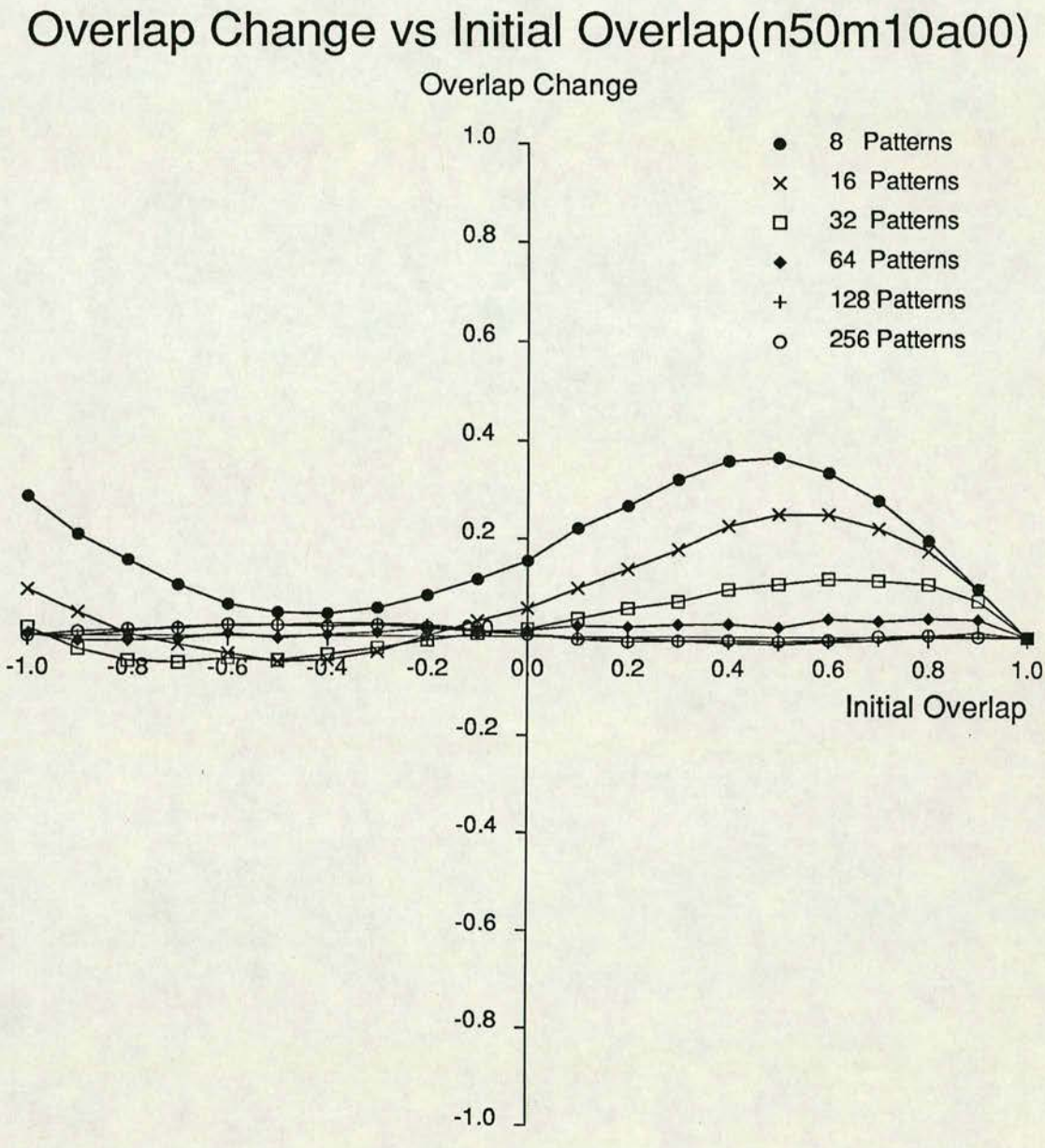

Figure 5.2: Overlap change for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 1$.

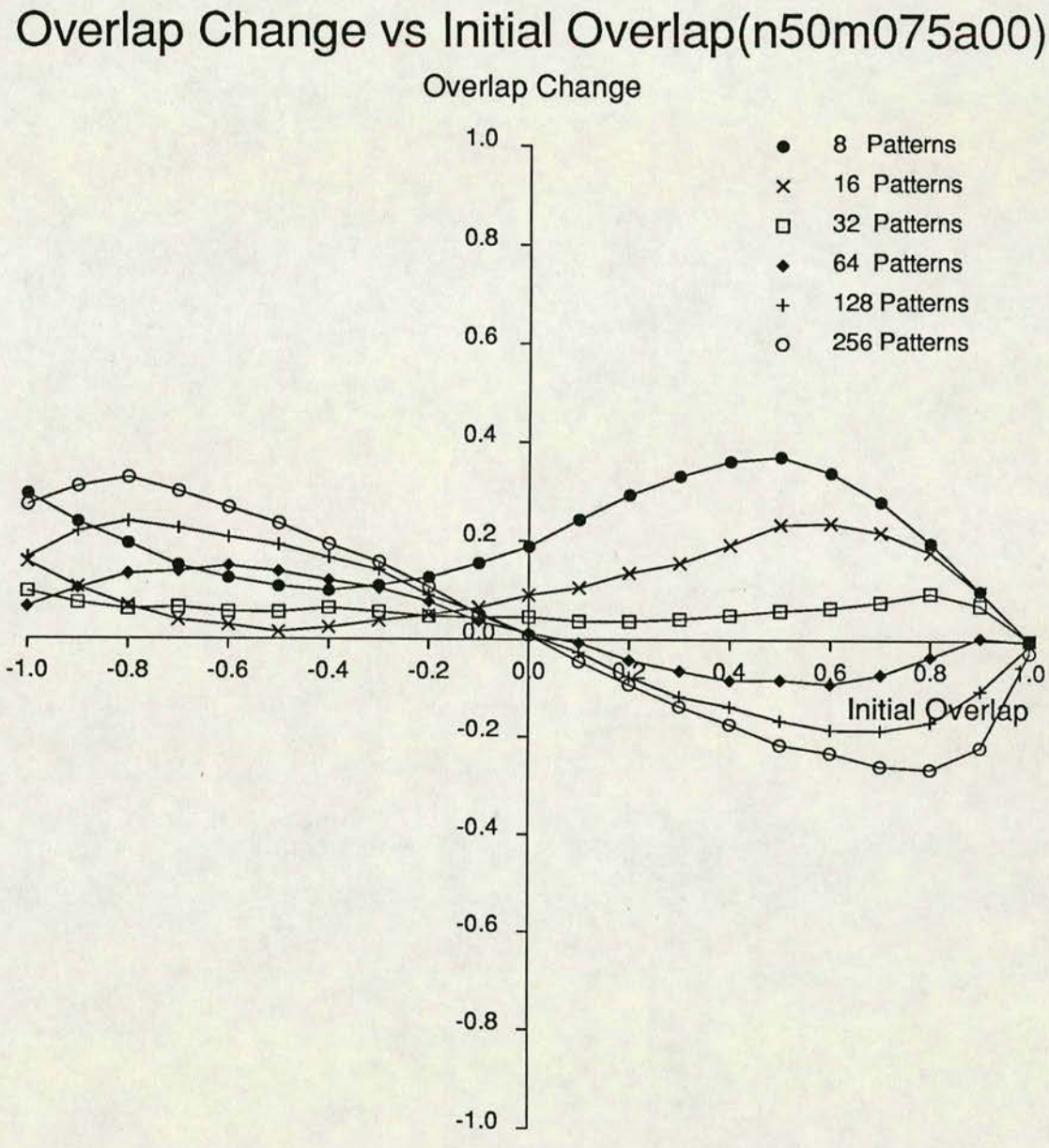

Figure 5.3: Overlap change for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0.75$.

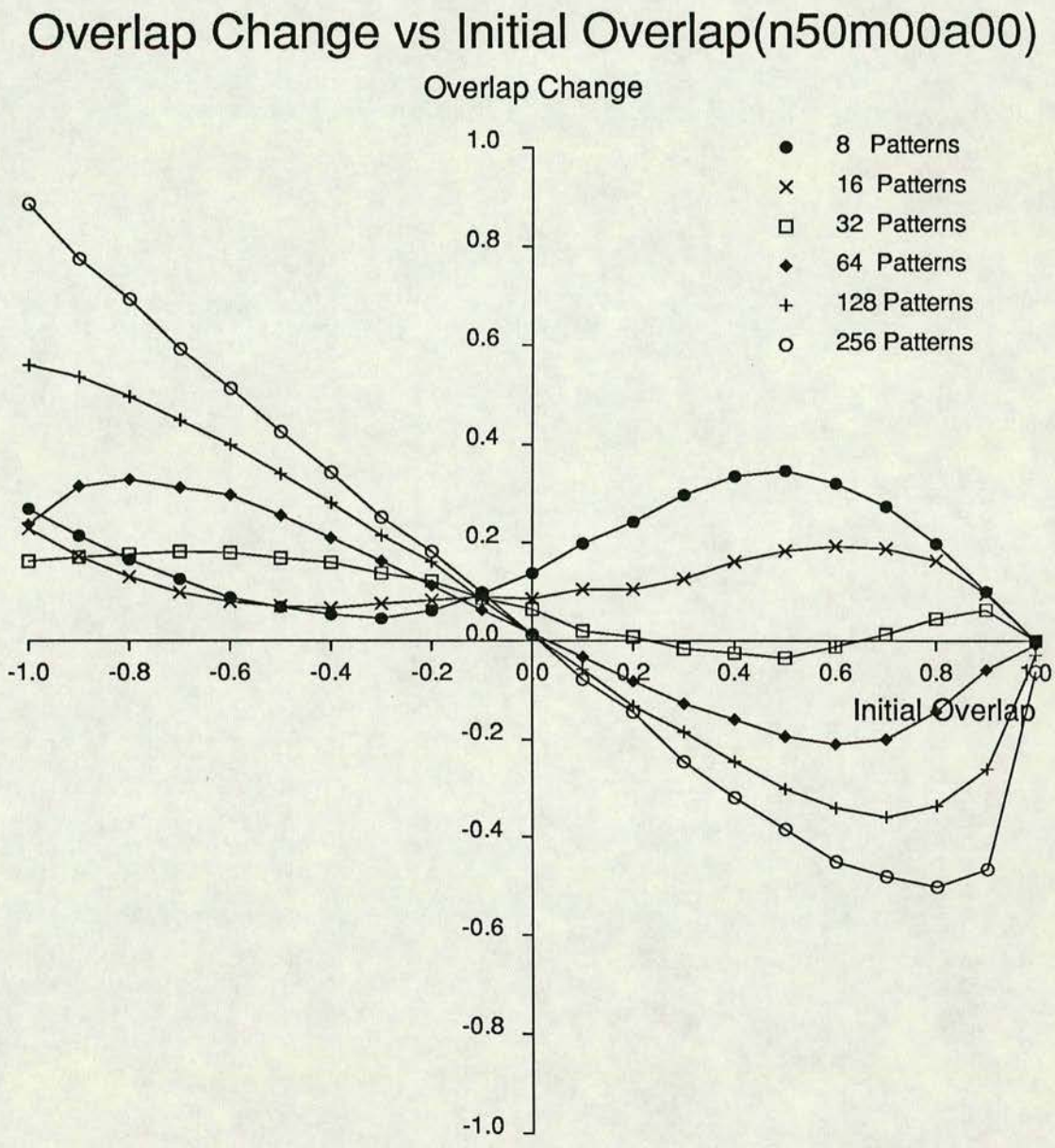

Figure 5.4: Overlap change for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0$.

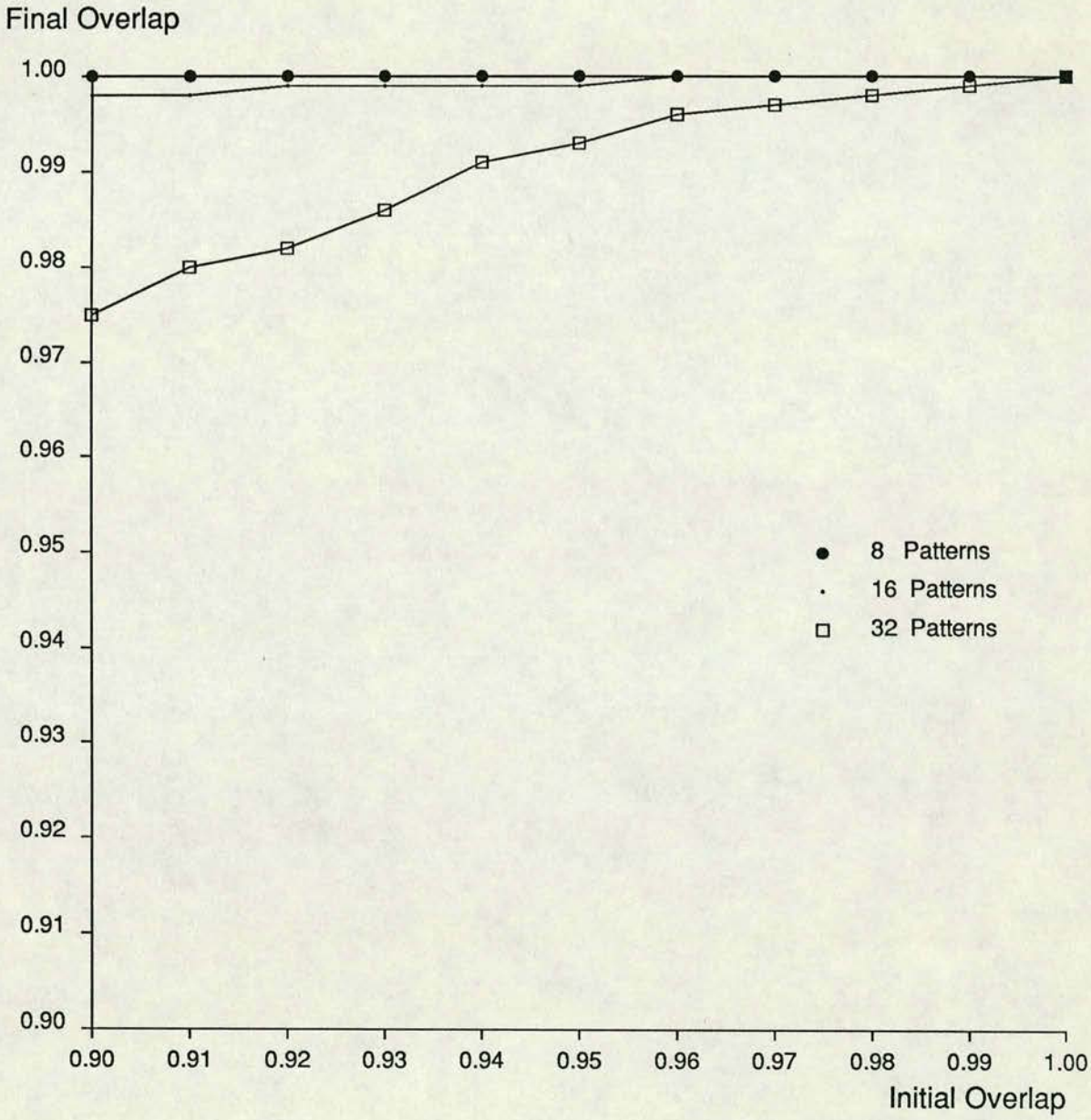

Figure 5.5: Information required to store a pattern for various pattern magnetisation and pattern-target overlaps.

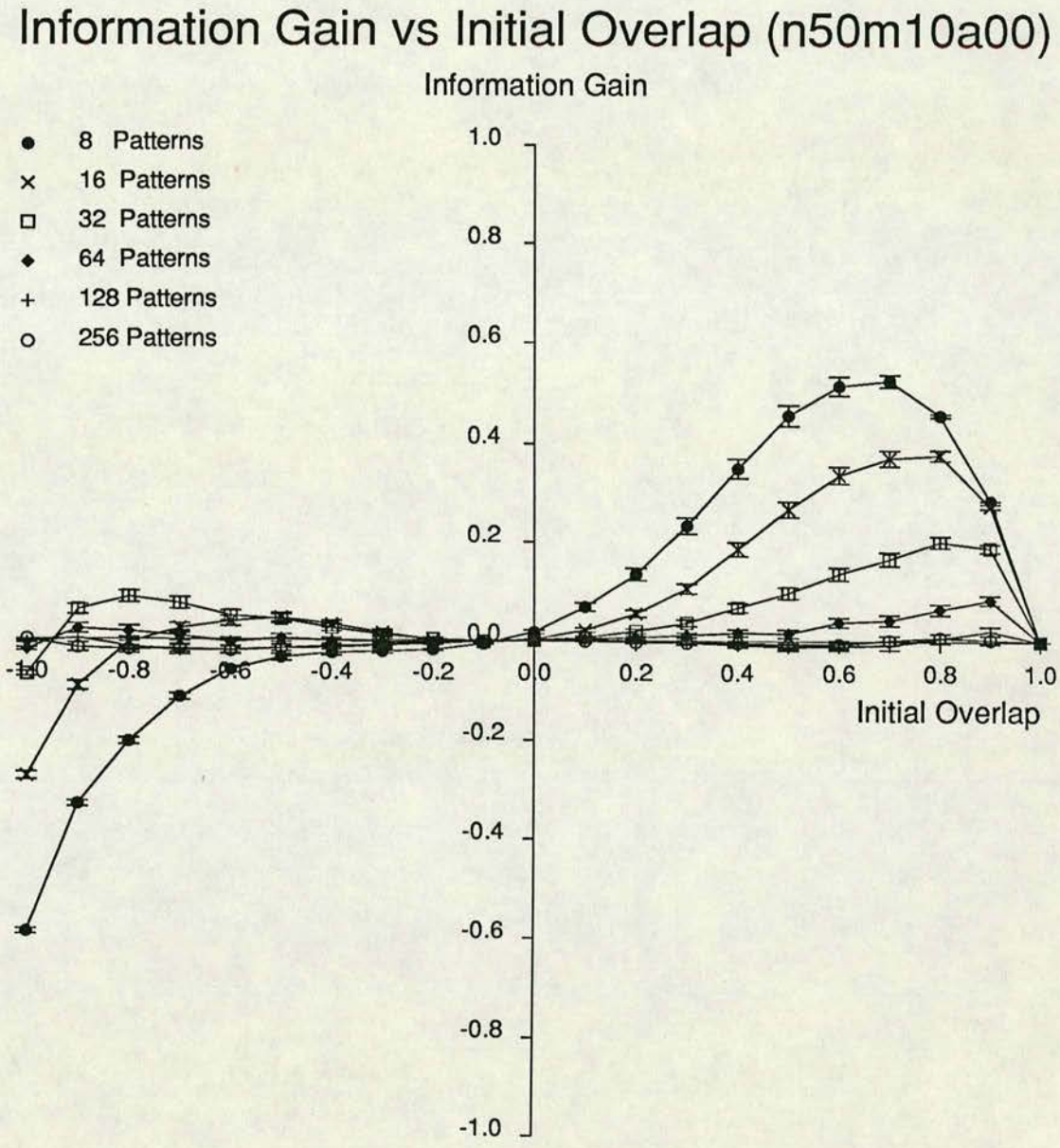

Figure 5.6: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 1$.

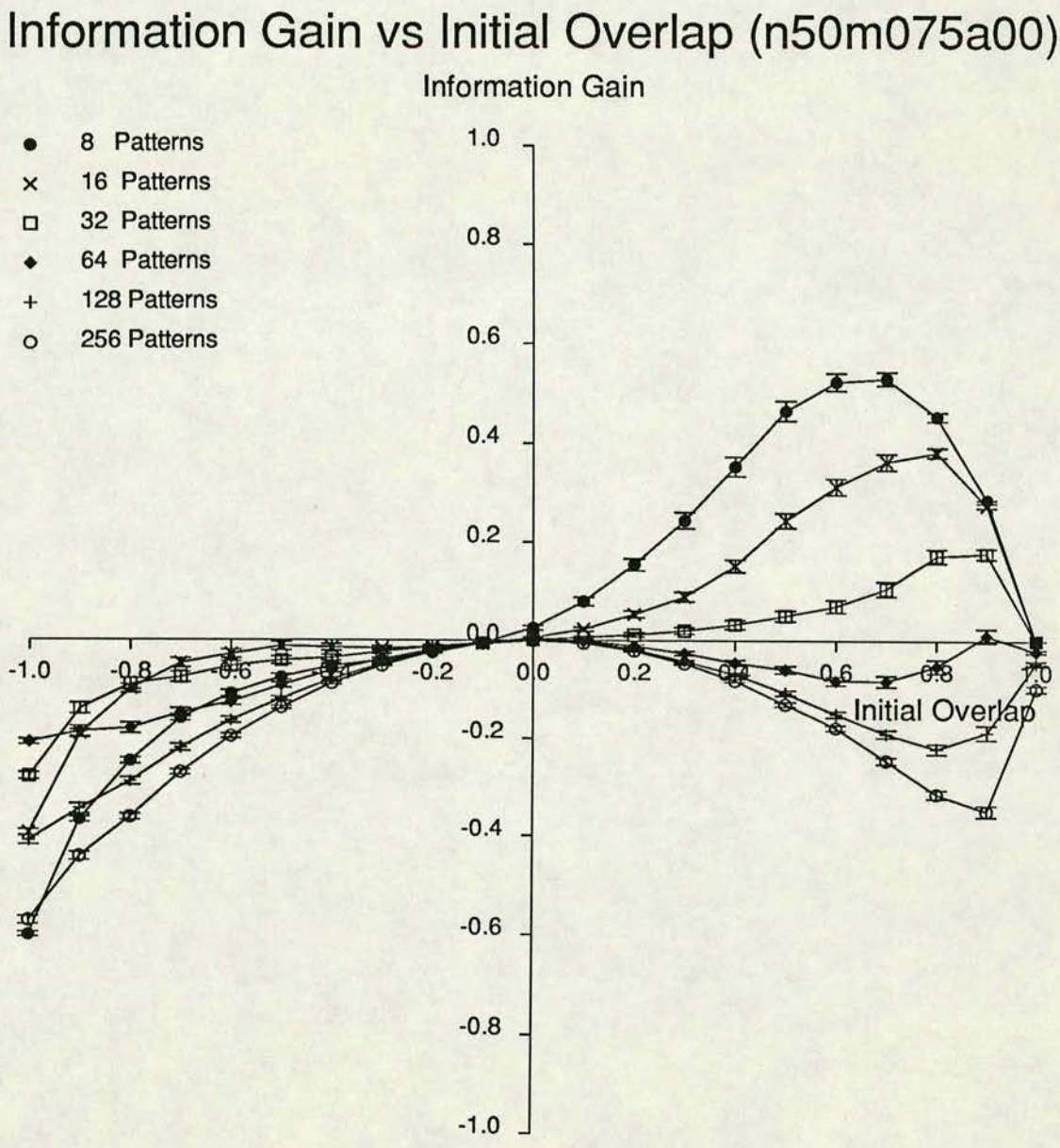

Figure 5.7: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0.75$.

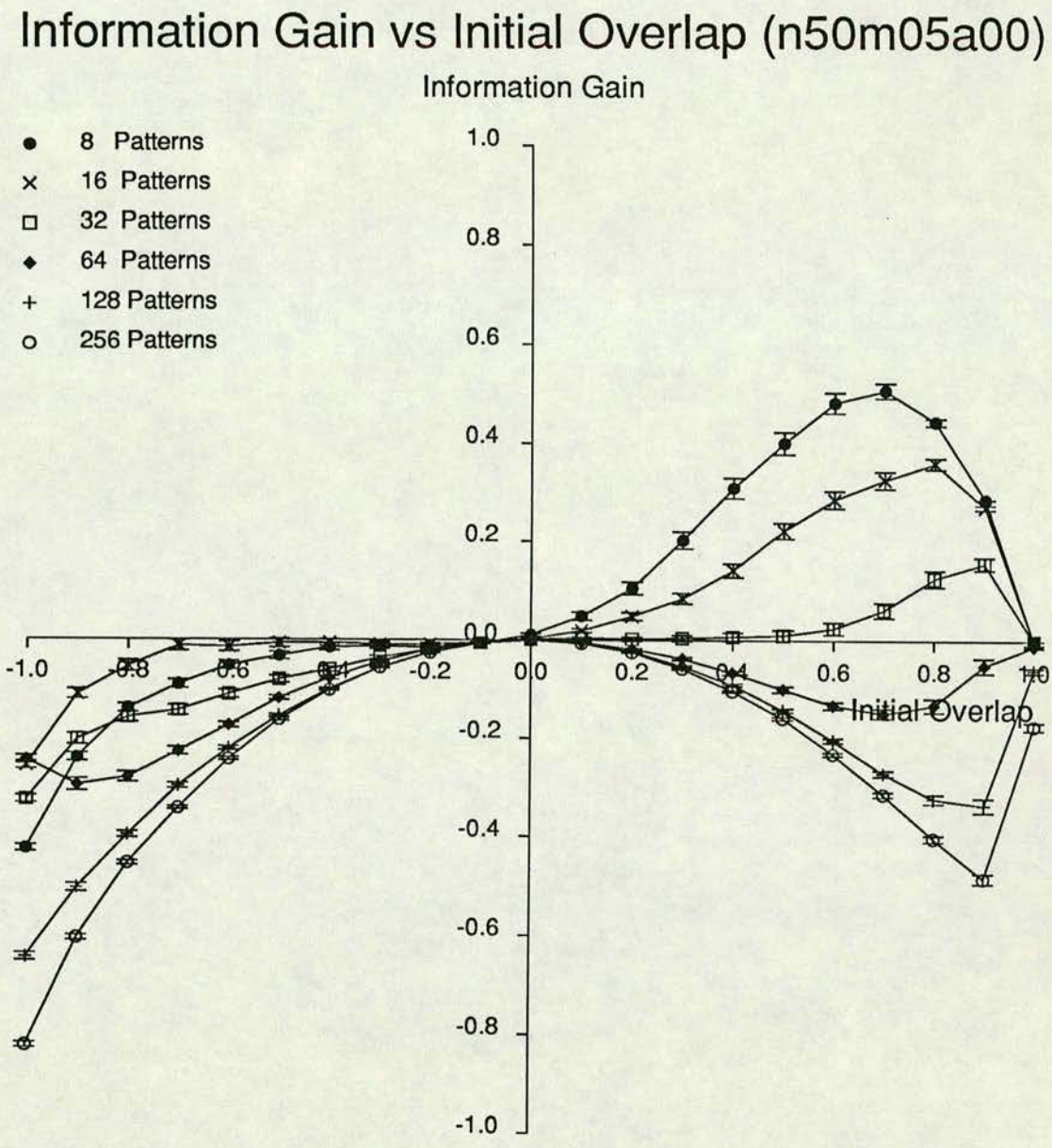

Figure 5.8: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0.5$.

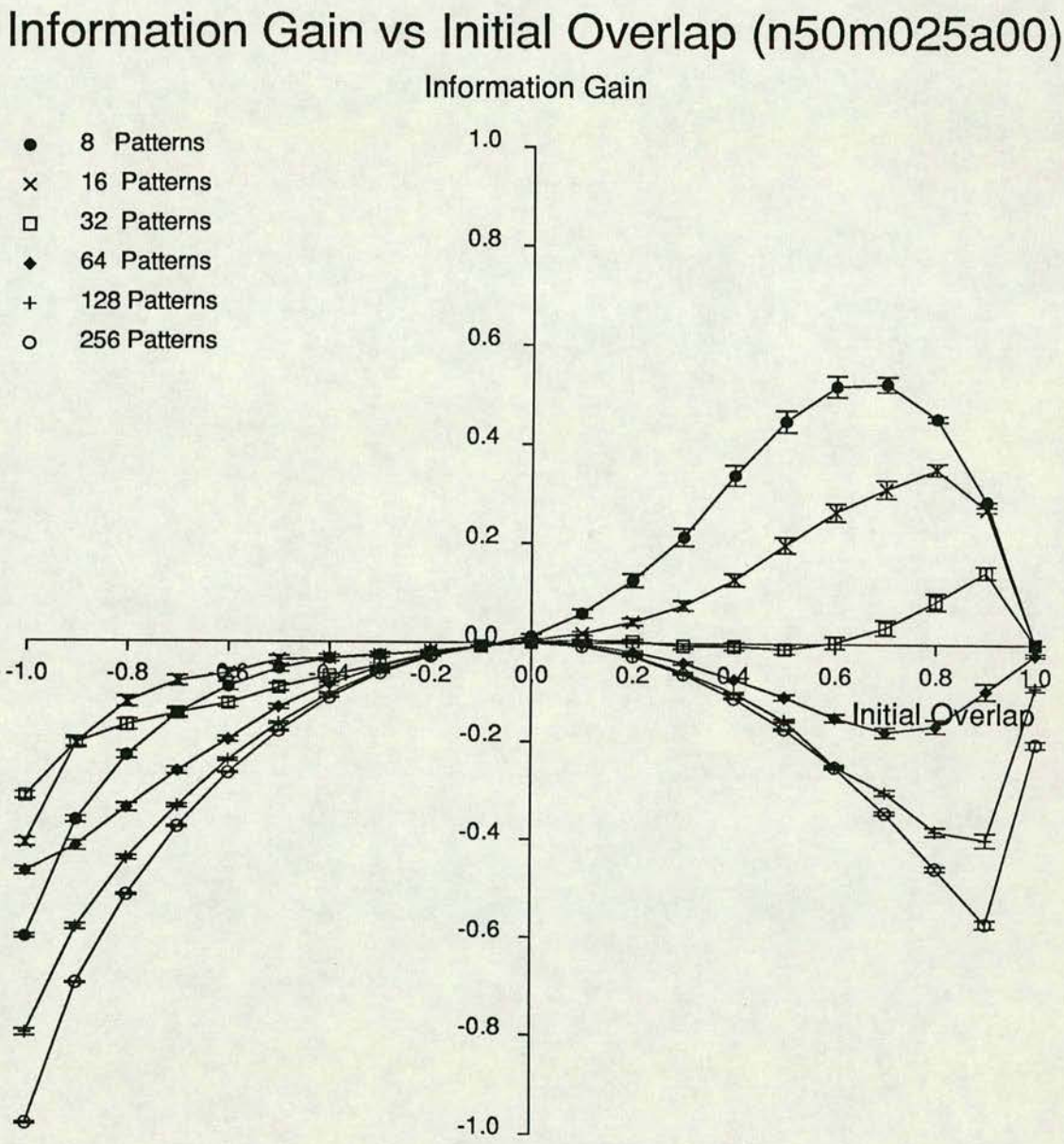

Figure 5.9: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0.25$.

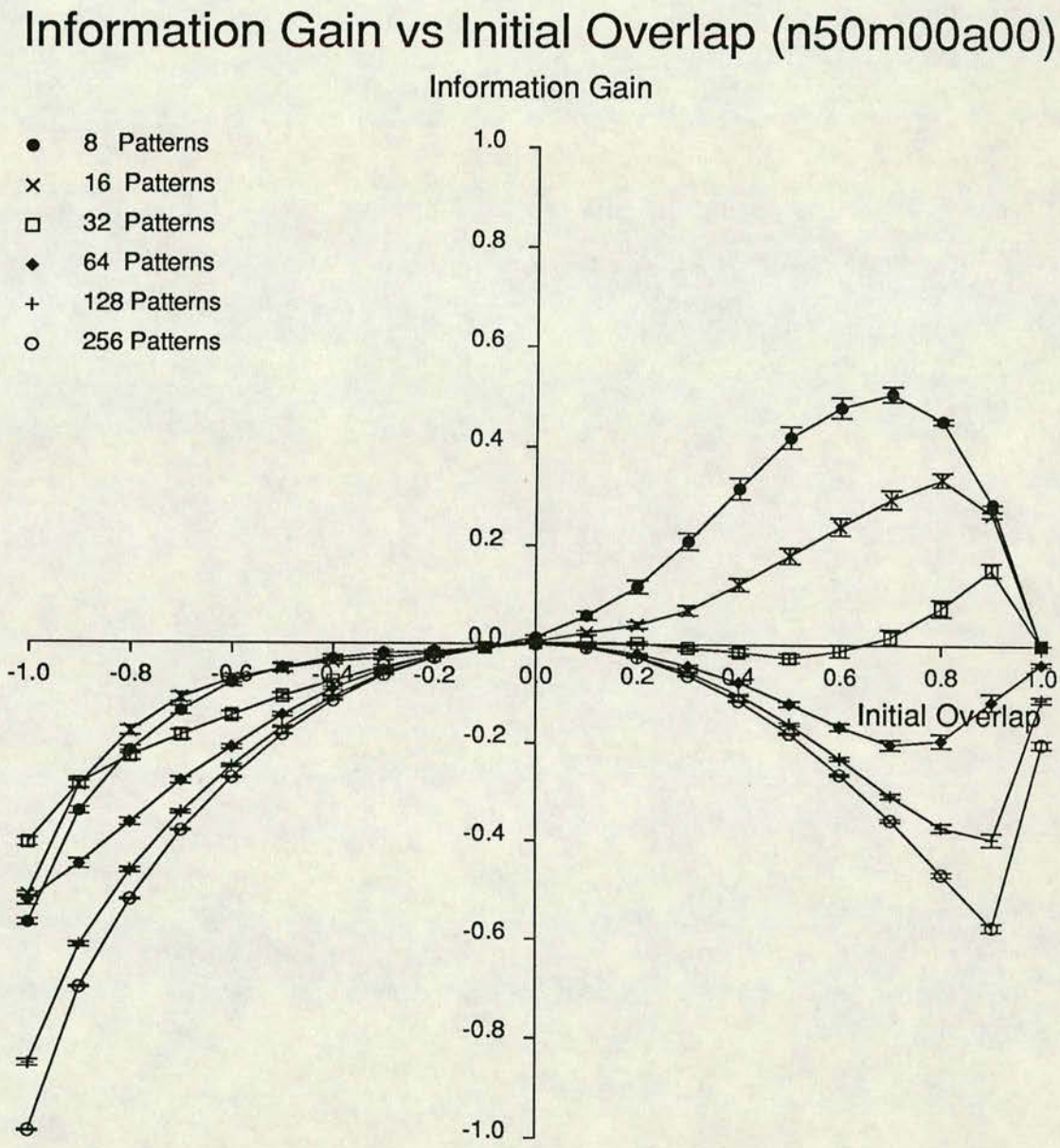

Figure 5.10: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and input patterns. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0$.

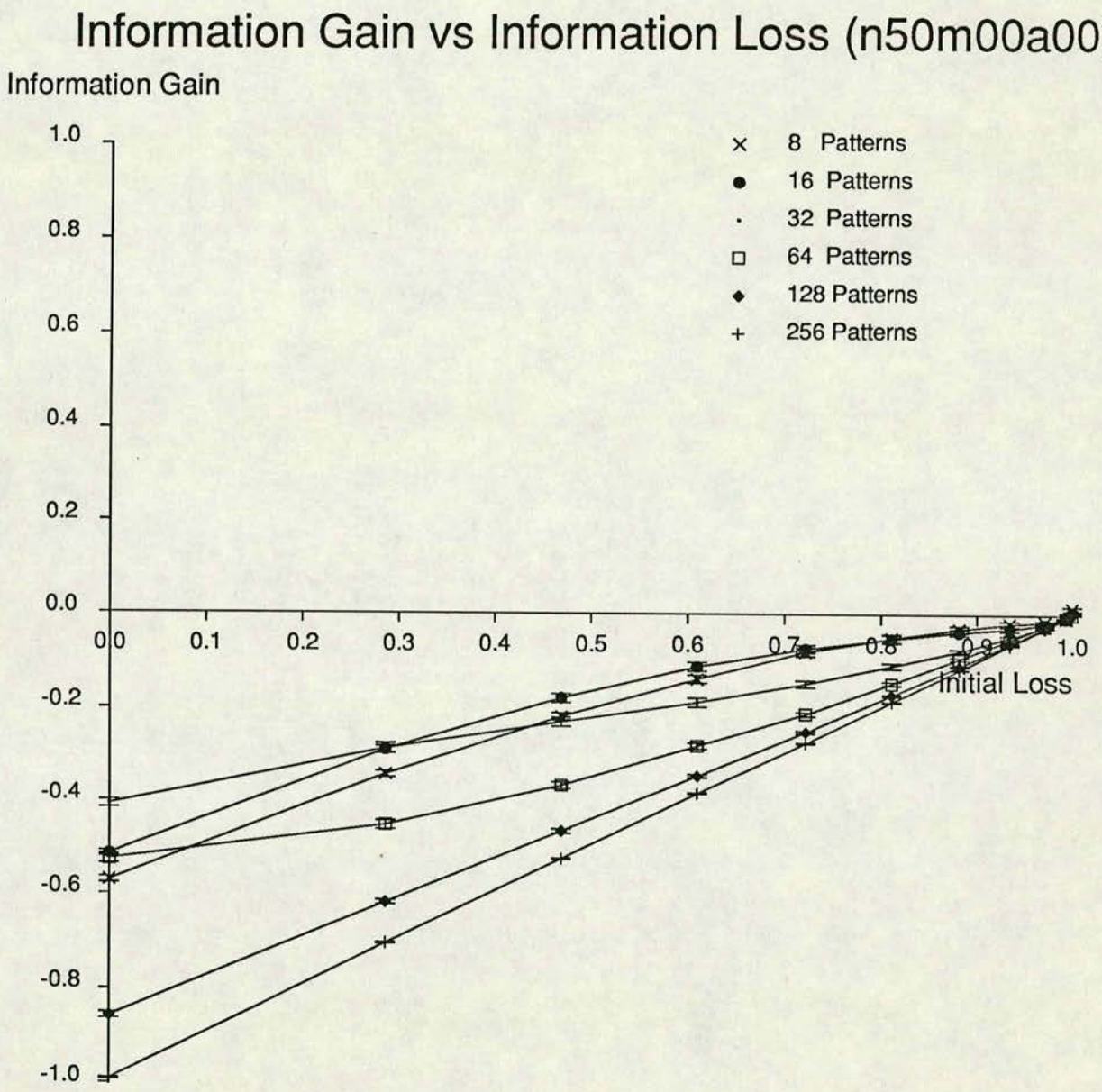

Figure 5.11: Information recovered from noise for various sizes of training set versus the information lost due to noise. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0$.

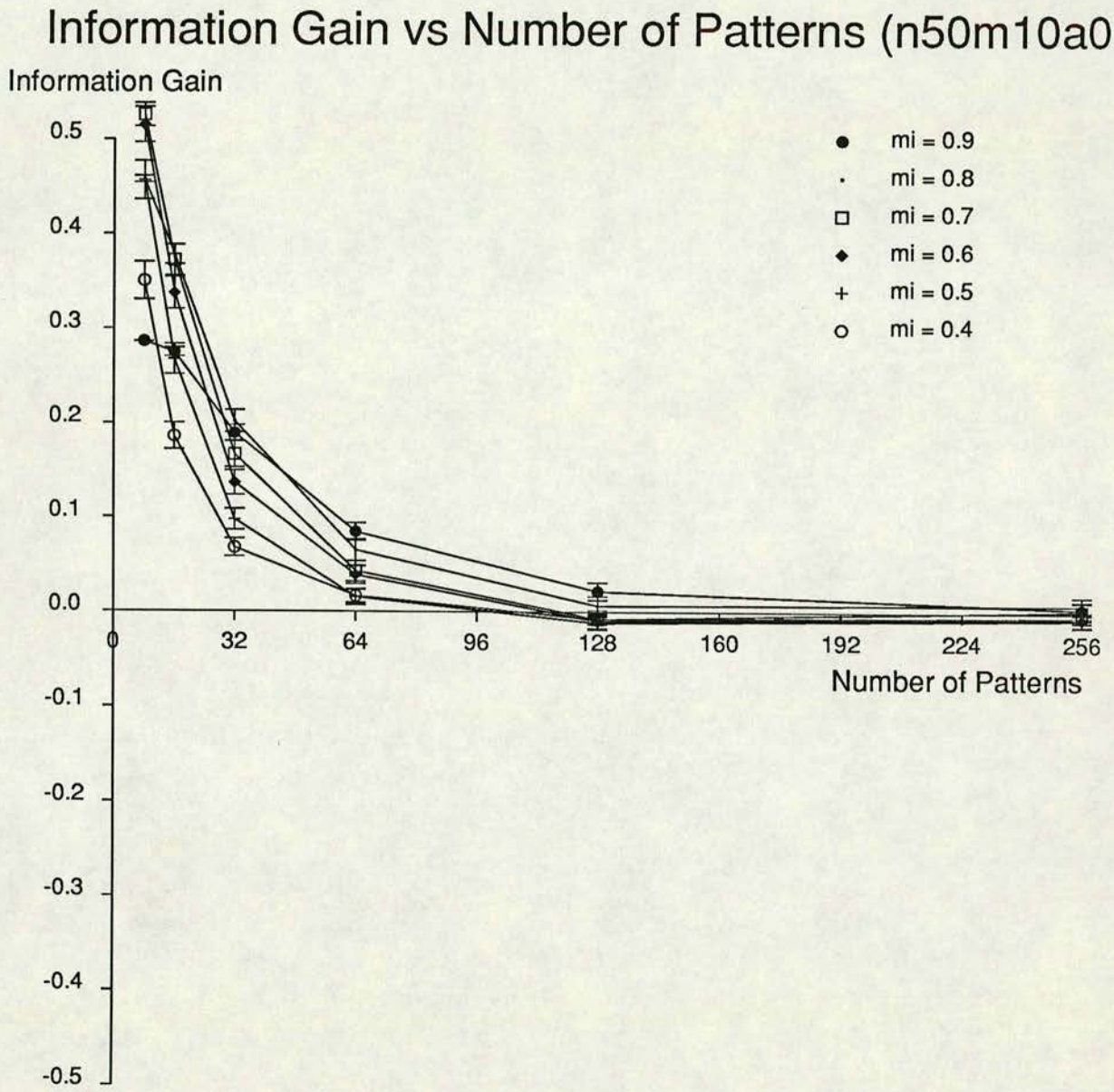

Figure 5.12: Information recovered from noise for initial overlaps versus the number of patterns stored. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 1$.

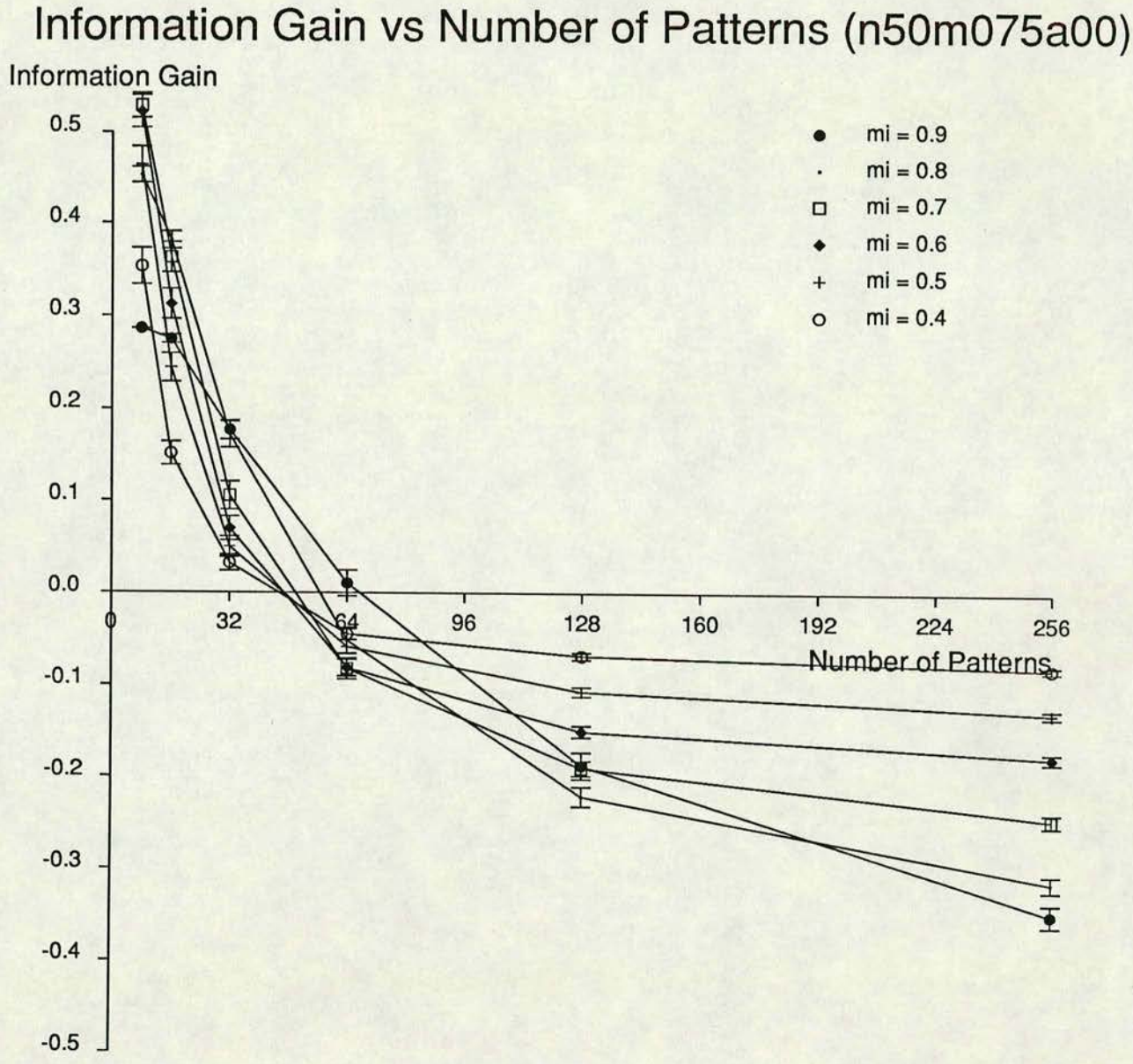

Figure 5.13: Information recovered from noise for initial overlaps versus the number of patterns stored. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0.75$.

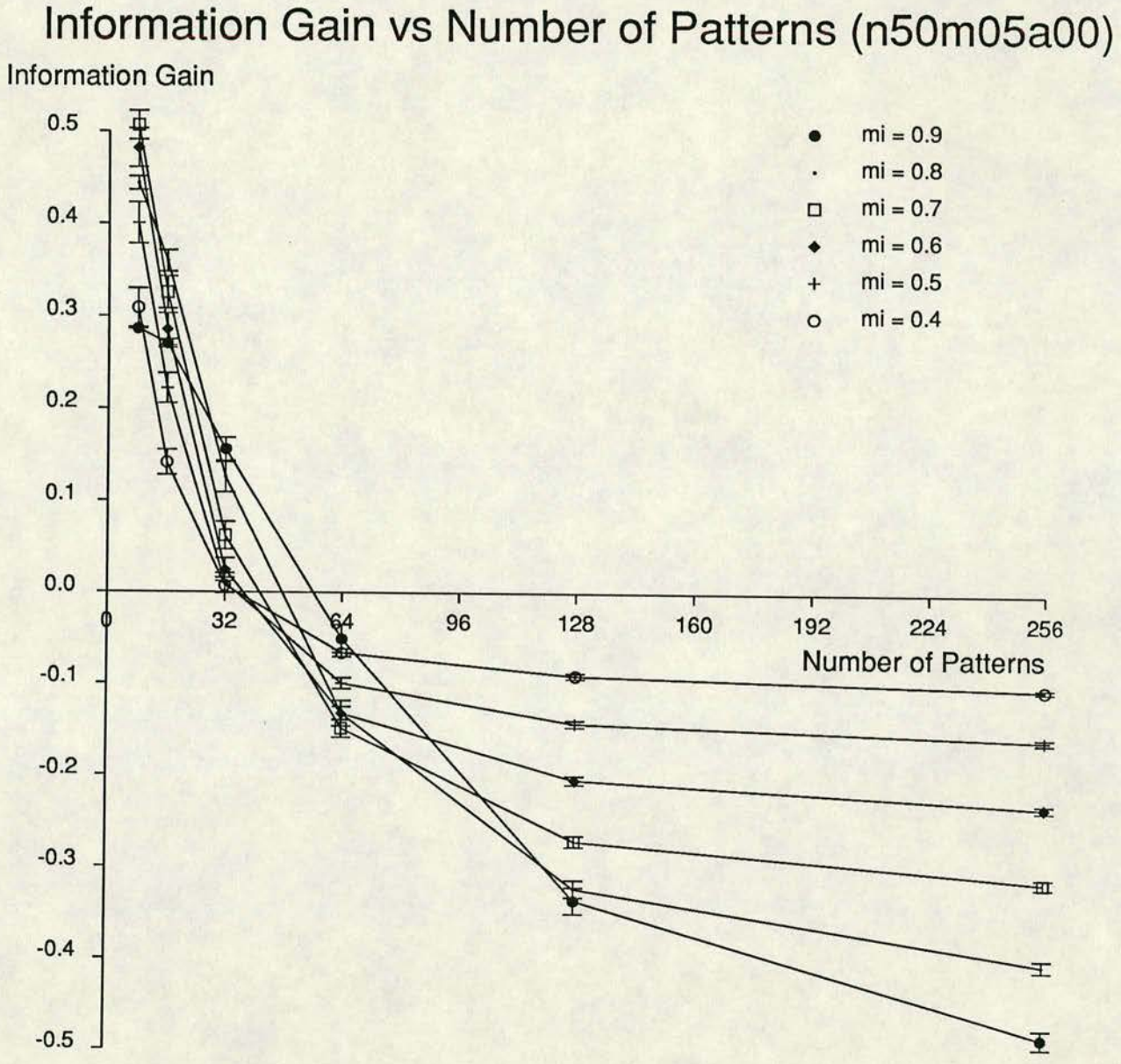

Figure 5.14: Information recovered from noise for initial overlaps versus the number of patterns stored. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0.5$.

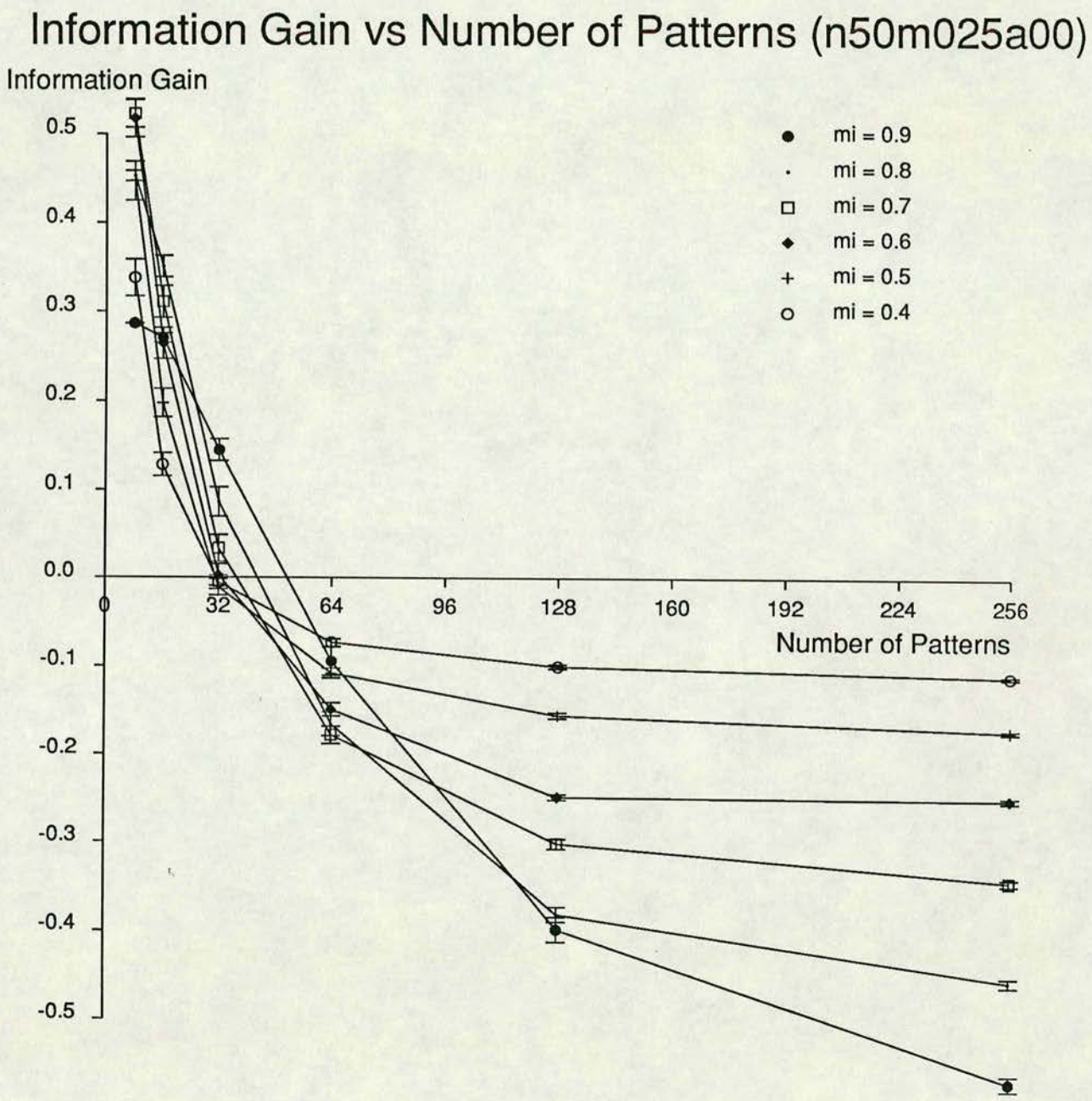

Figure 5.15: Information recovered from noise for initial overlaps versus the number of patterns stored. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0.25$.

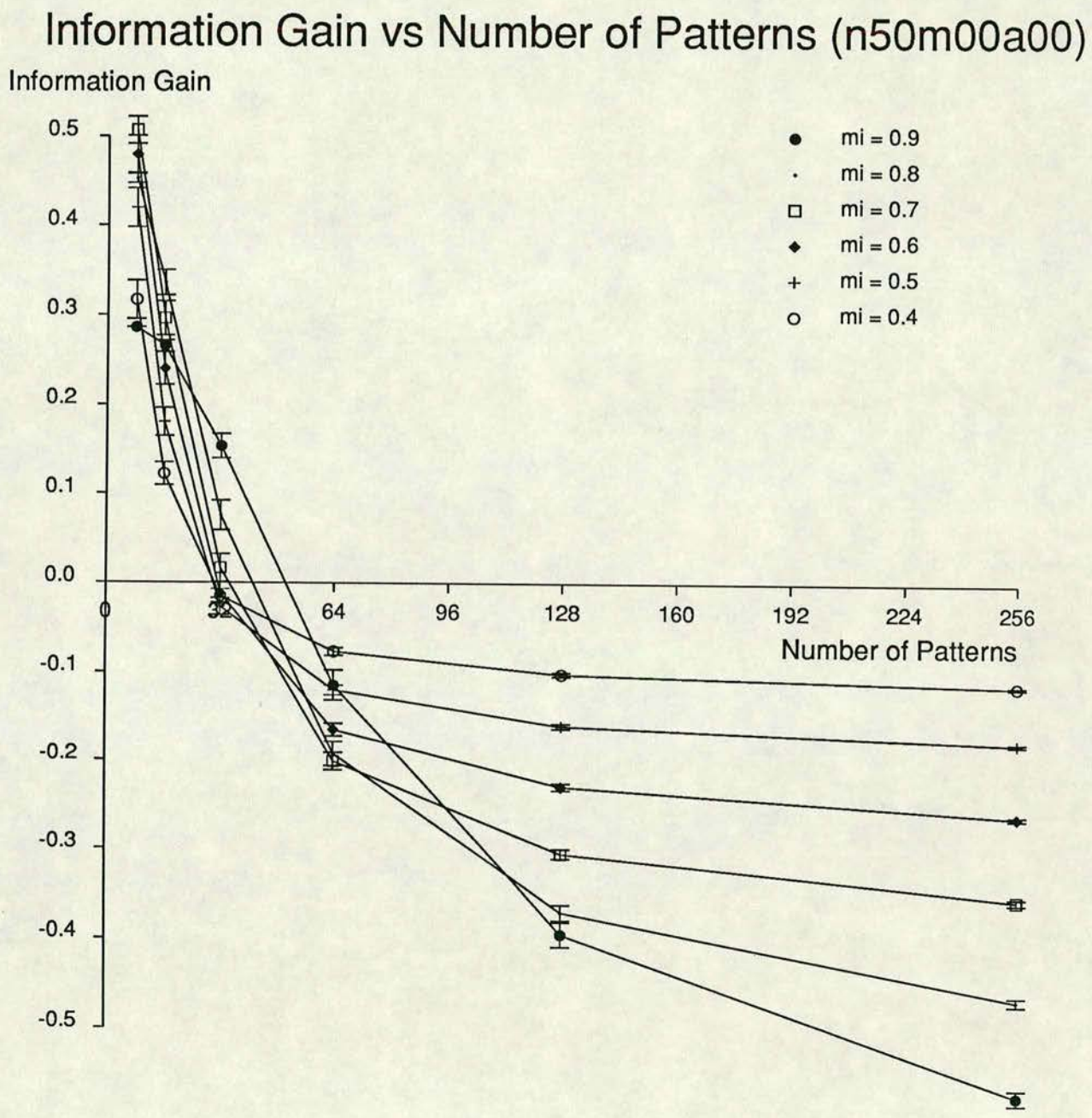

Figure 5.16: Information recovered from noise for initial overlaps versus the number of patterns stored. The network has fifty hidden nodes and is trained on unbiased patterns with $m_p = 0$.

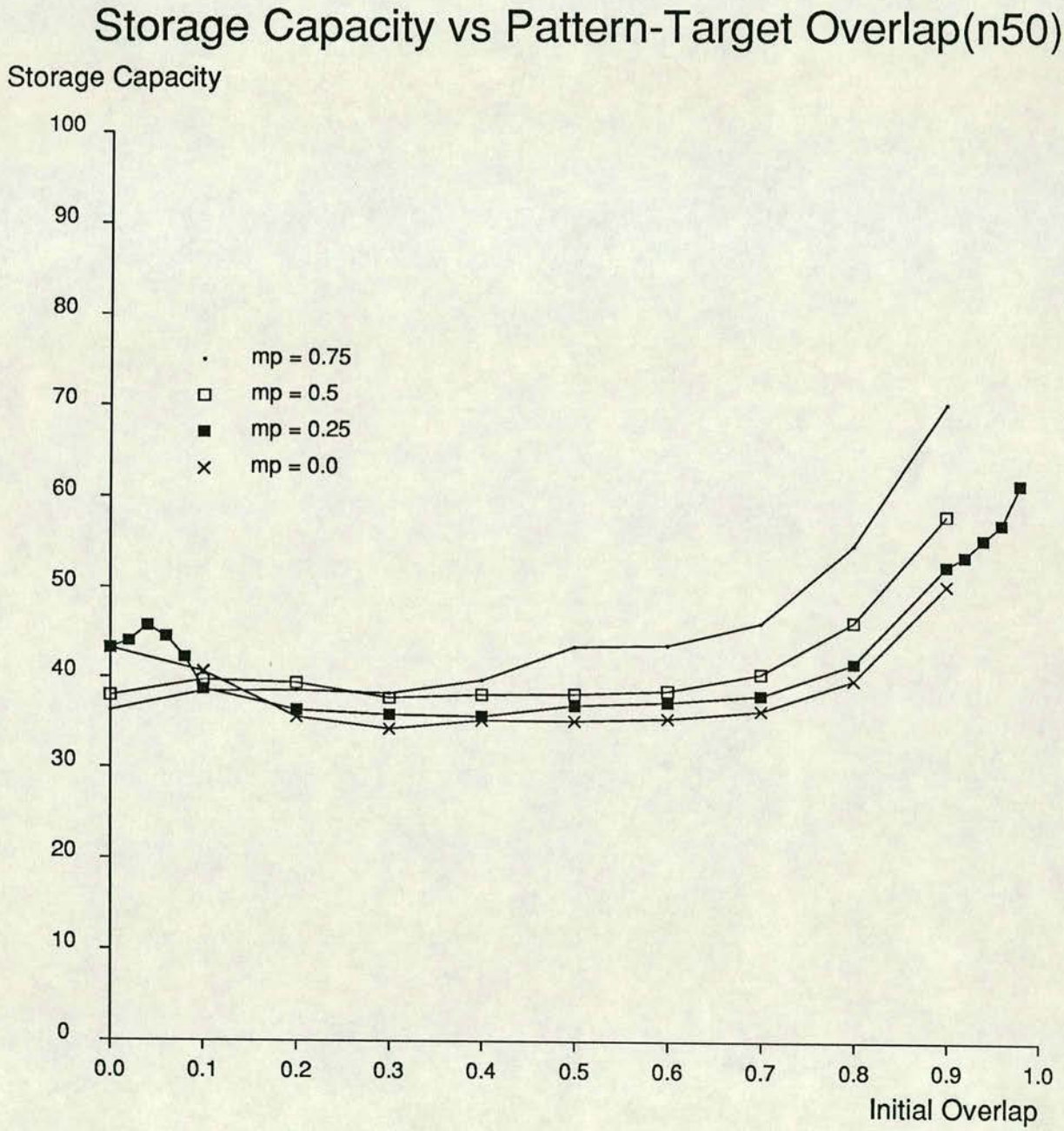

Figure 5.17: Intercept point versus initial overlap for fifty hidden nodes.

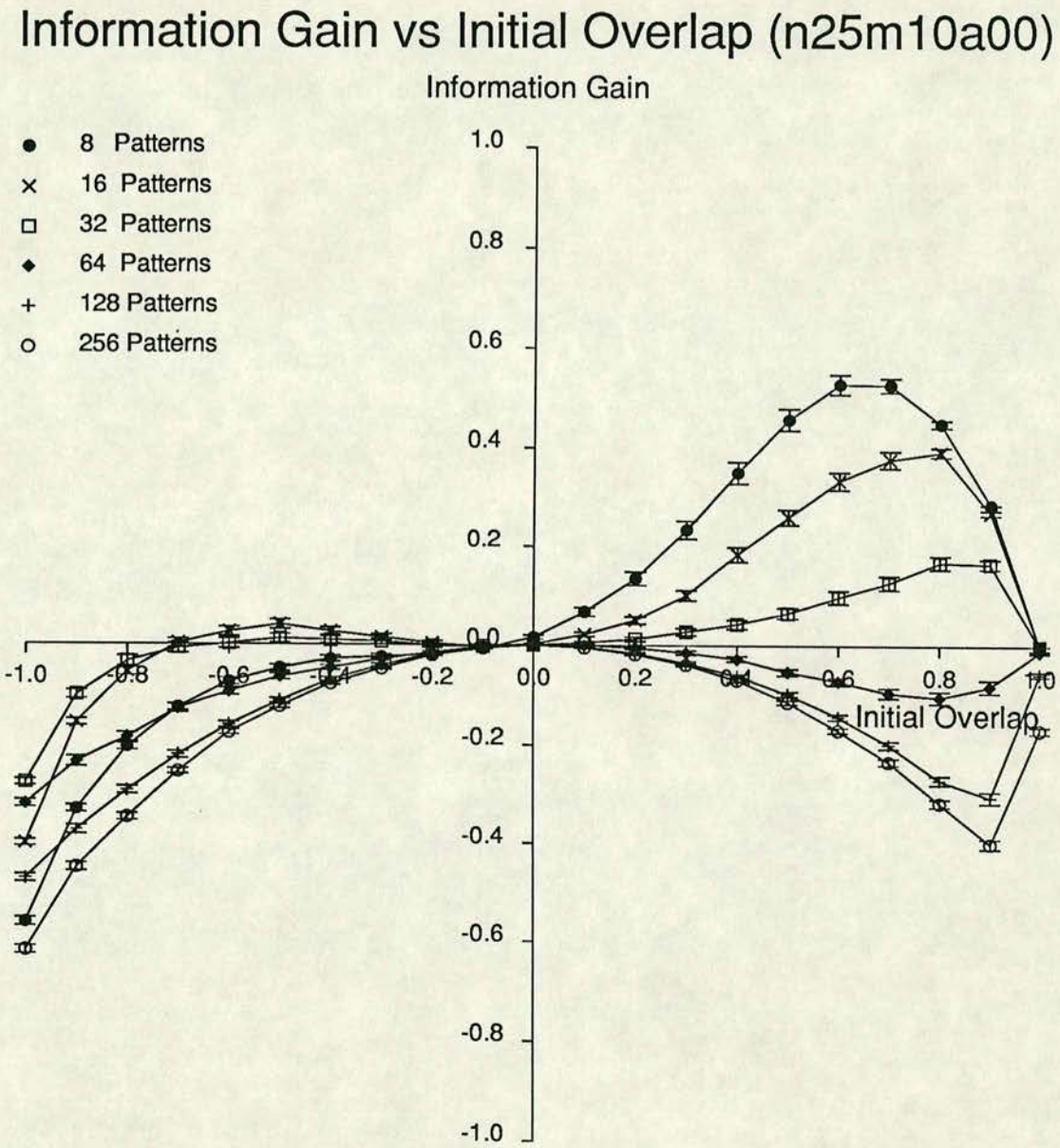

Figure 5.18: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has twenty-five hidden nodes and is trained on unbiased patterns with $m_p = 1$.

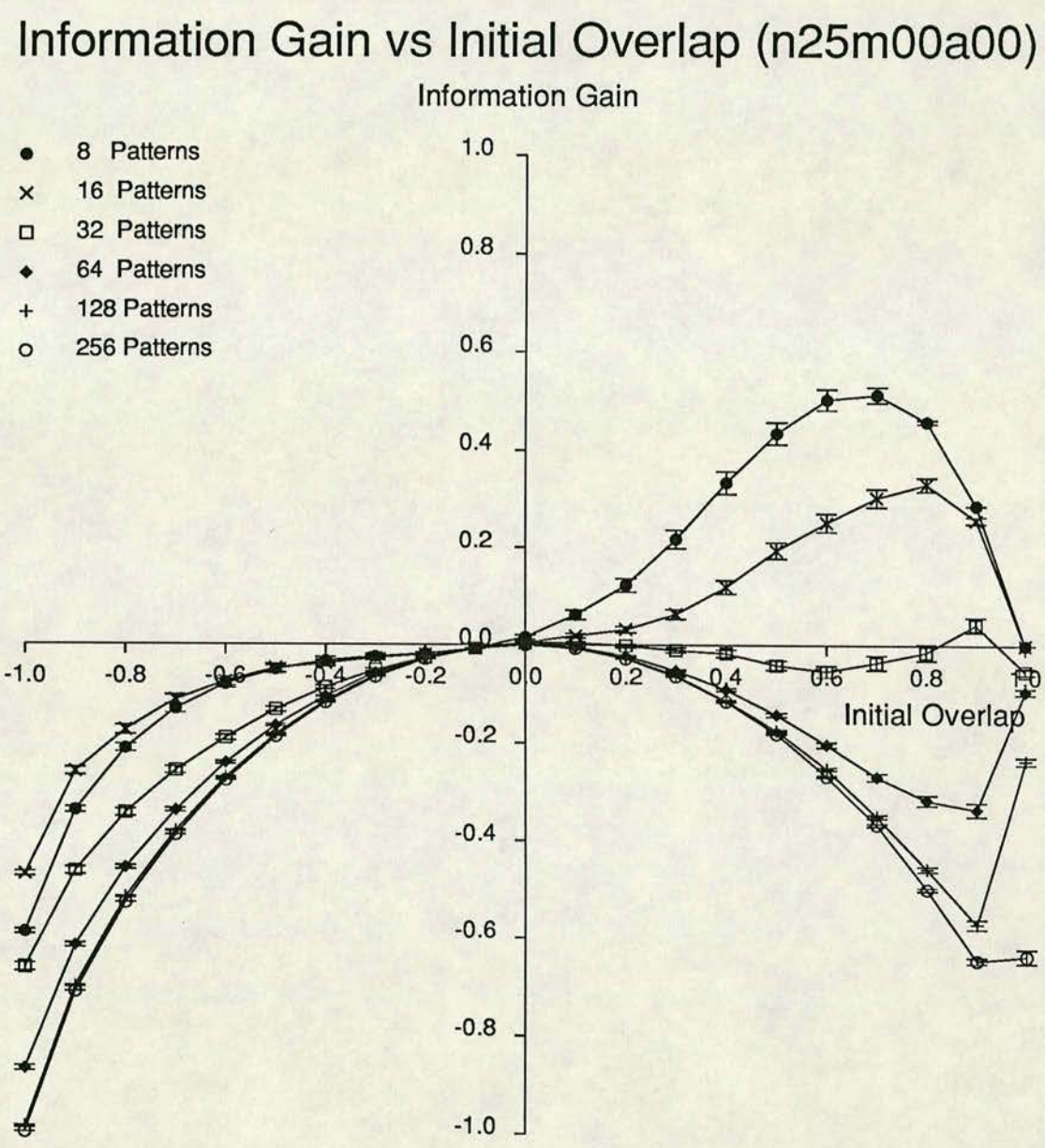

Figure 5.19: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has twenty-five hidden nodes and is trained on unbiased patterns with $m_p = 0$.

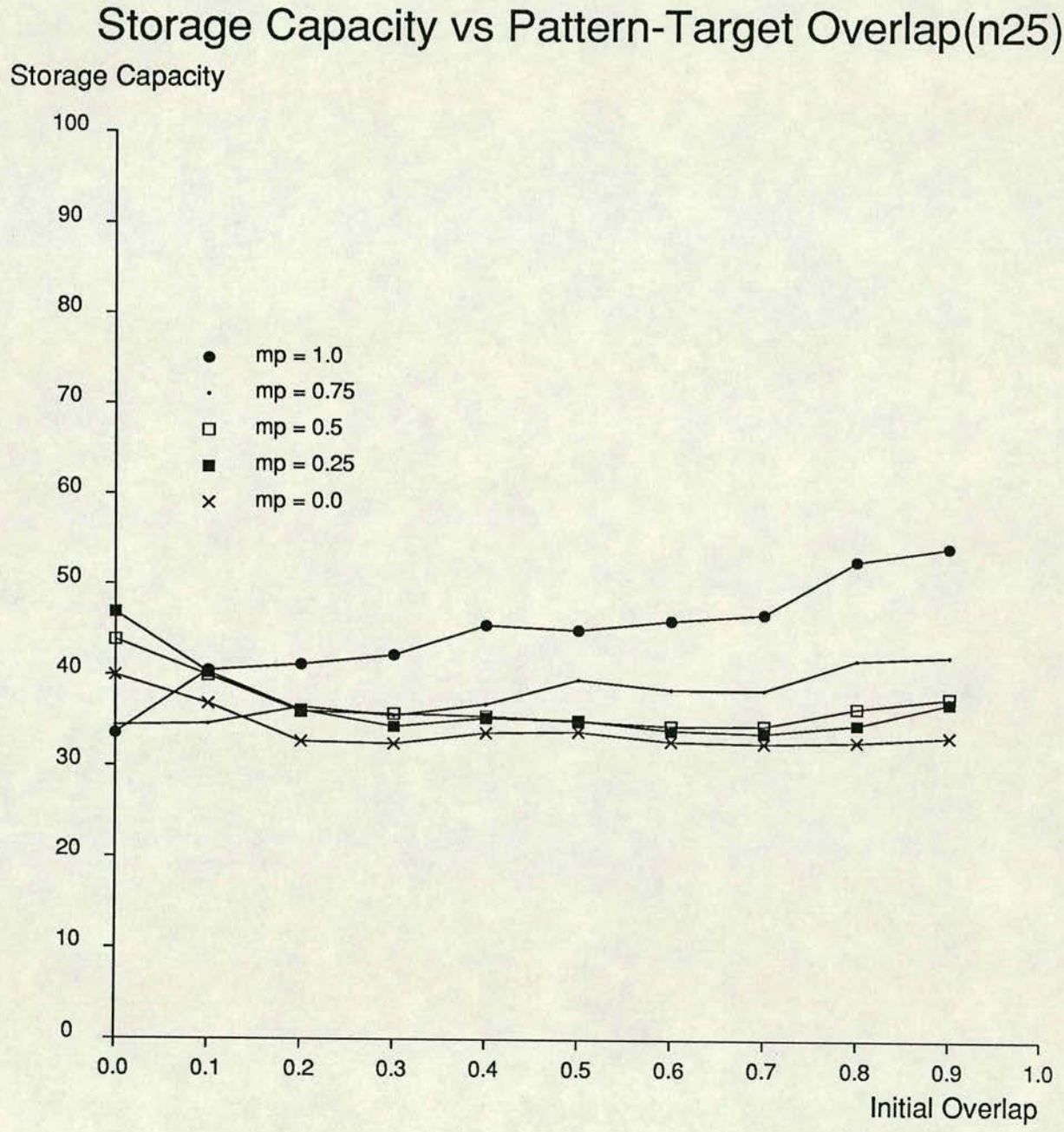

Figure 5.20: Intercept point versus initial overlap for twenty-five hidden nodes.

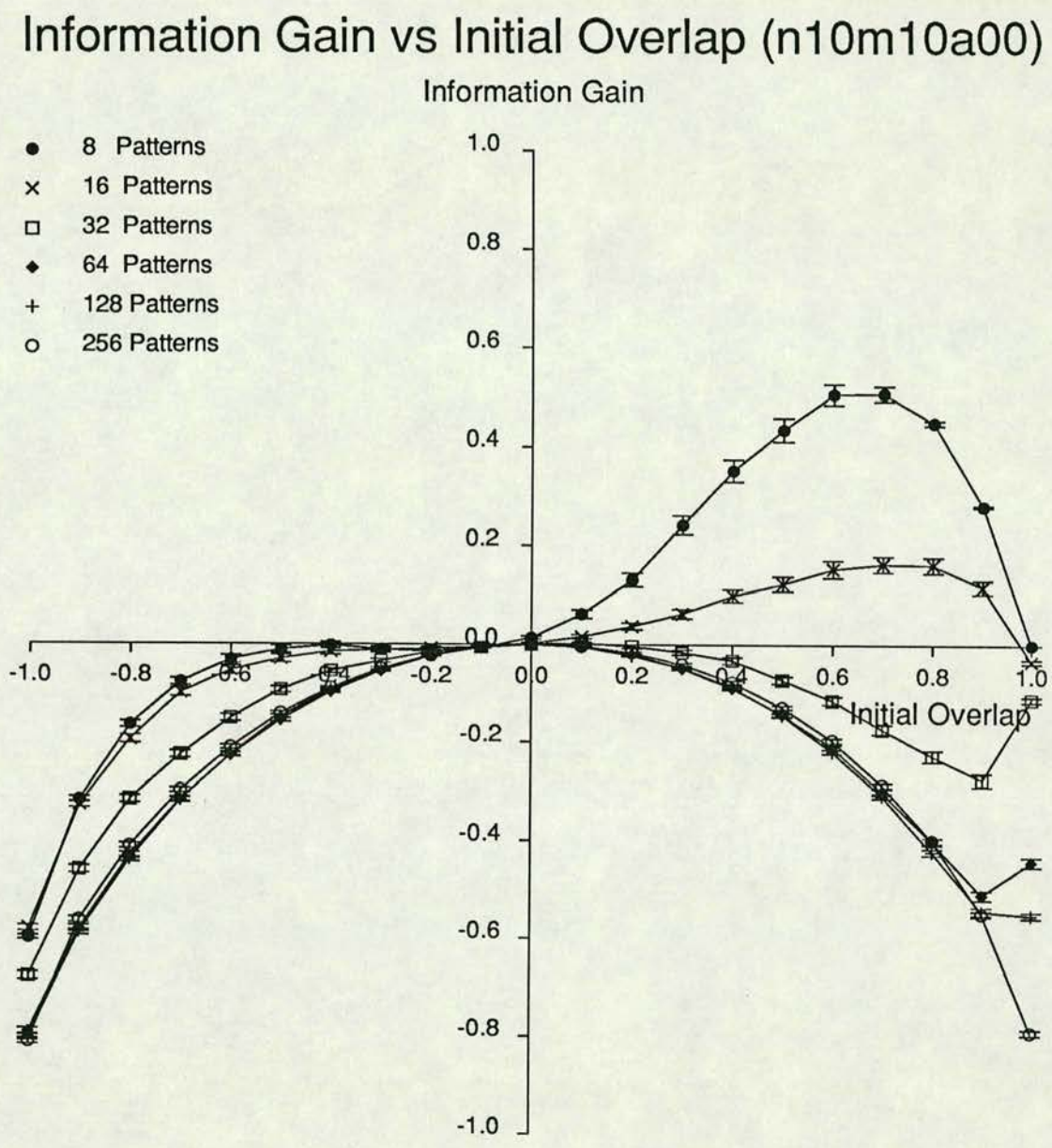

Figure 5.21: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has ten hidden nodes and is trained on unbiased patterns with $m_p = 1$.

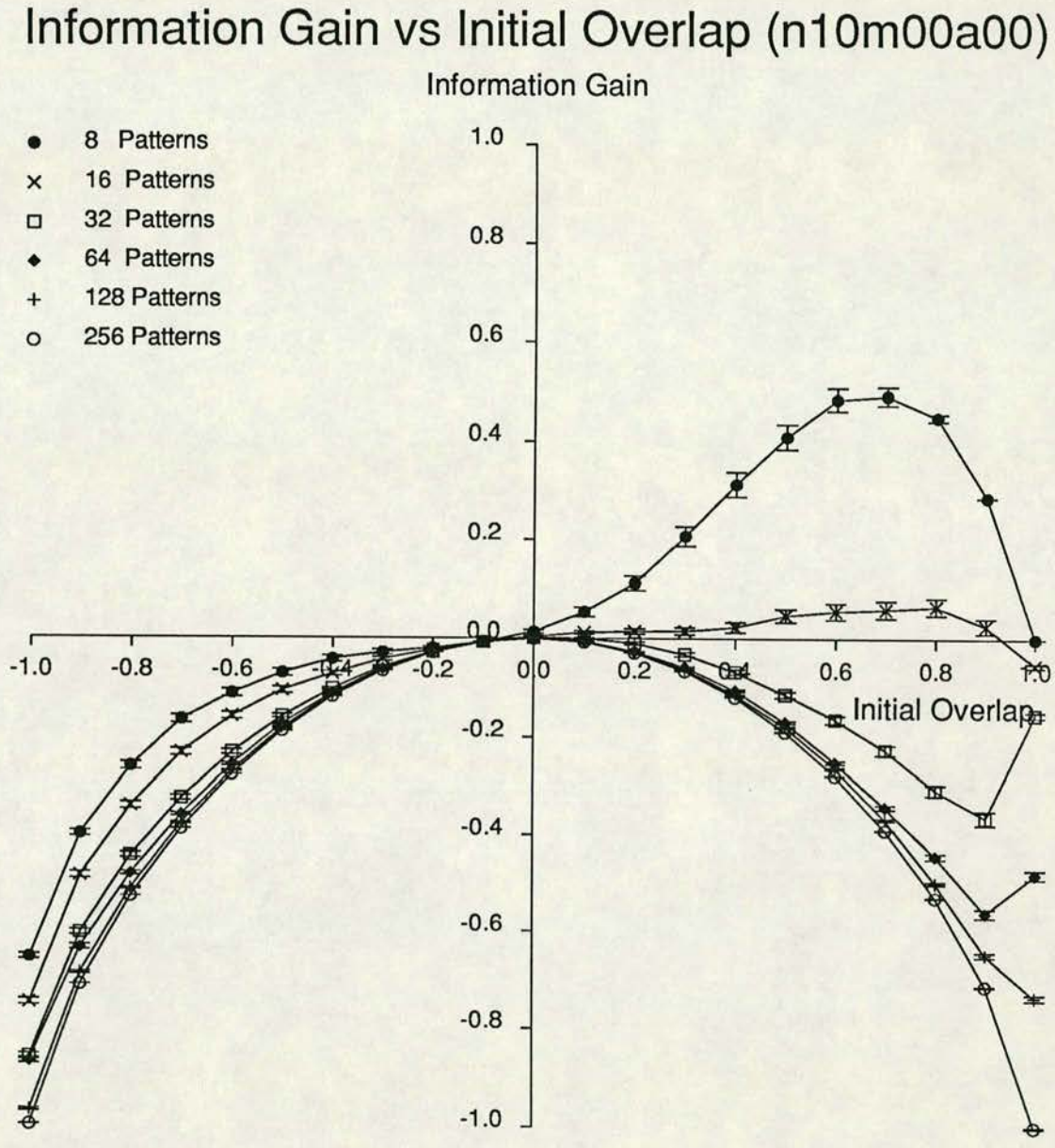

Figure 5.22: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has ten hidden nodes and is trained on unbiased patterns with $m_p = 0$.

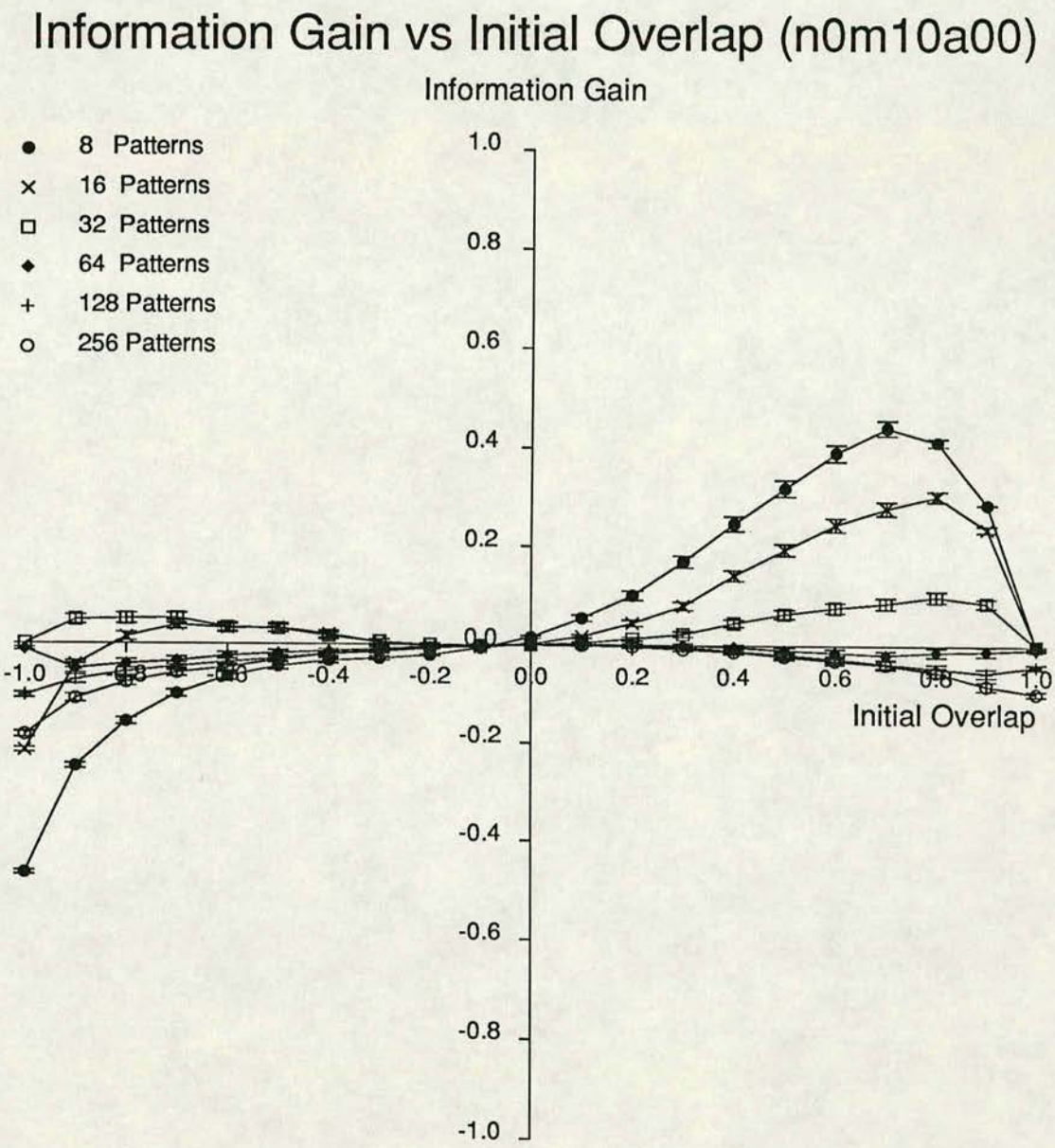

Figure 5.23: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has no hidden nodes and is trained on unbiased patterns with $m_p = 1$.

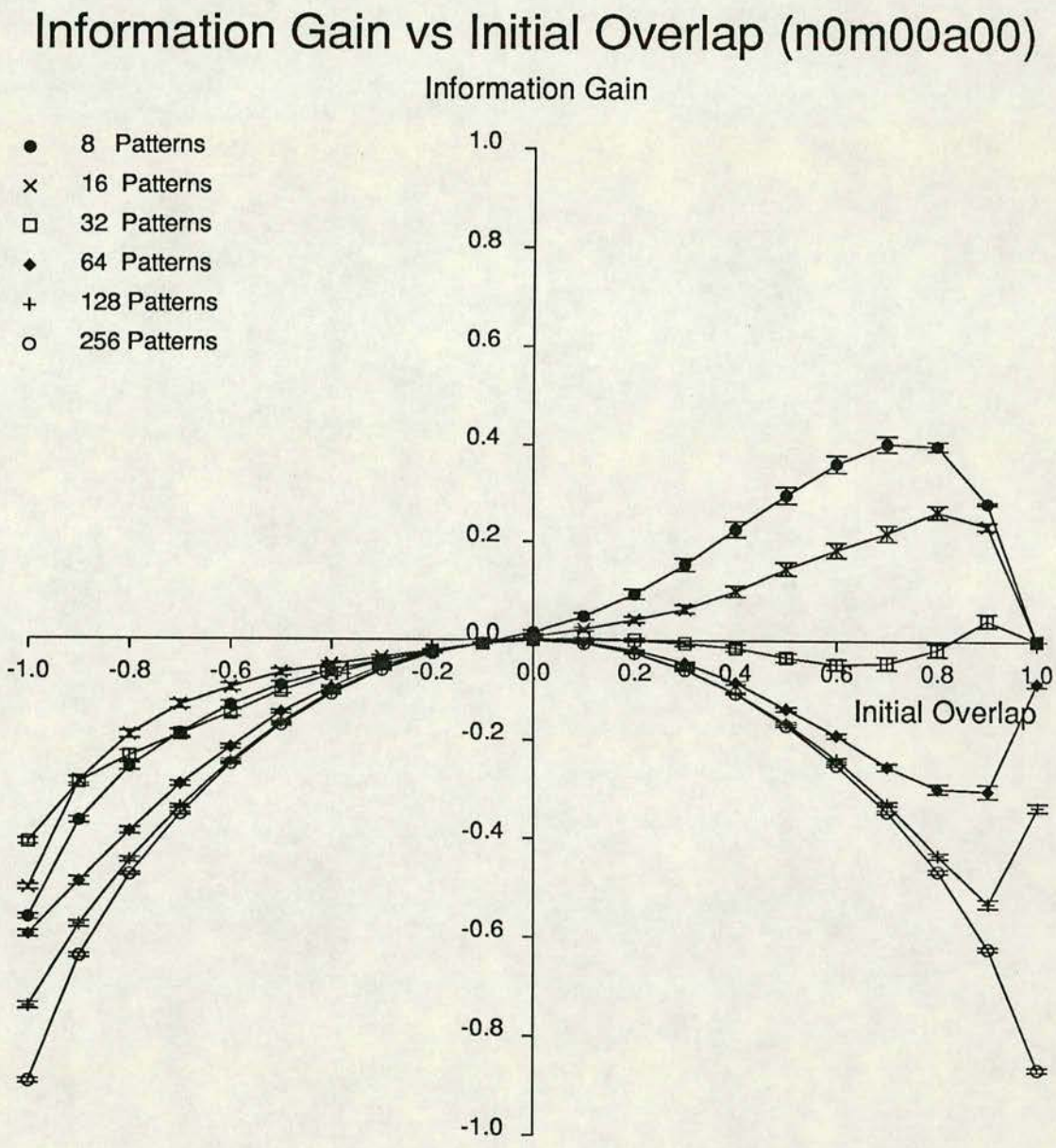

Figure 5.24: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has no hidden nodes and is trained on unbiased patterns with $m_p = 0$.

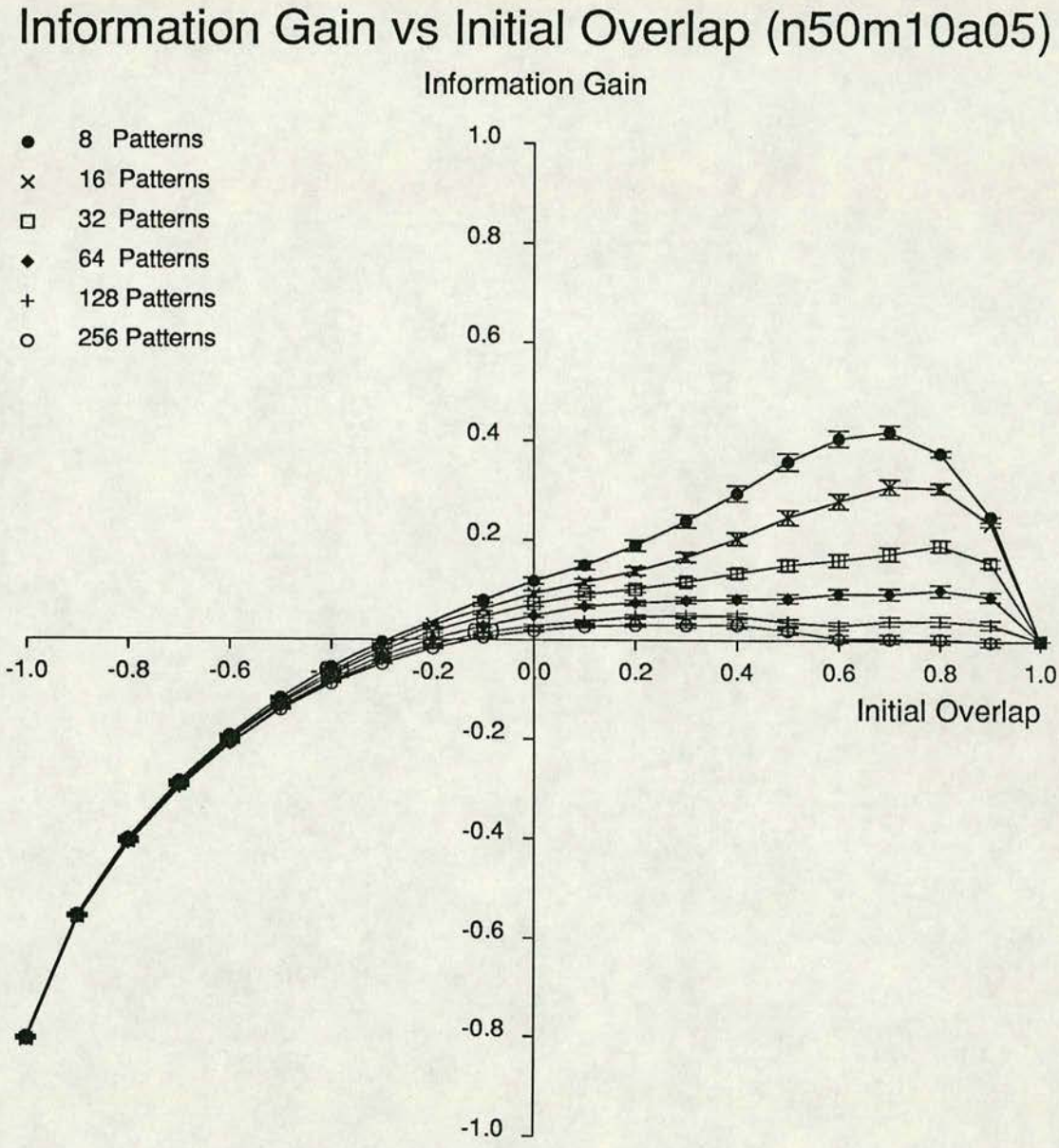

Figure 5.25: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has fifty hidden nodes and is trained on patterns with $a = 0.5$ and $m_p = 0$.

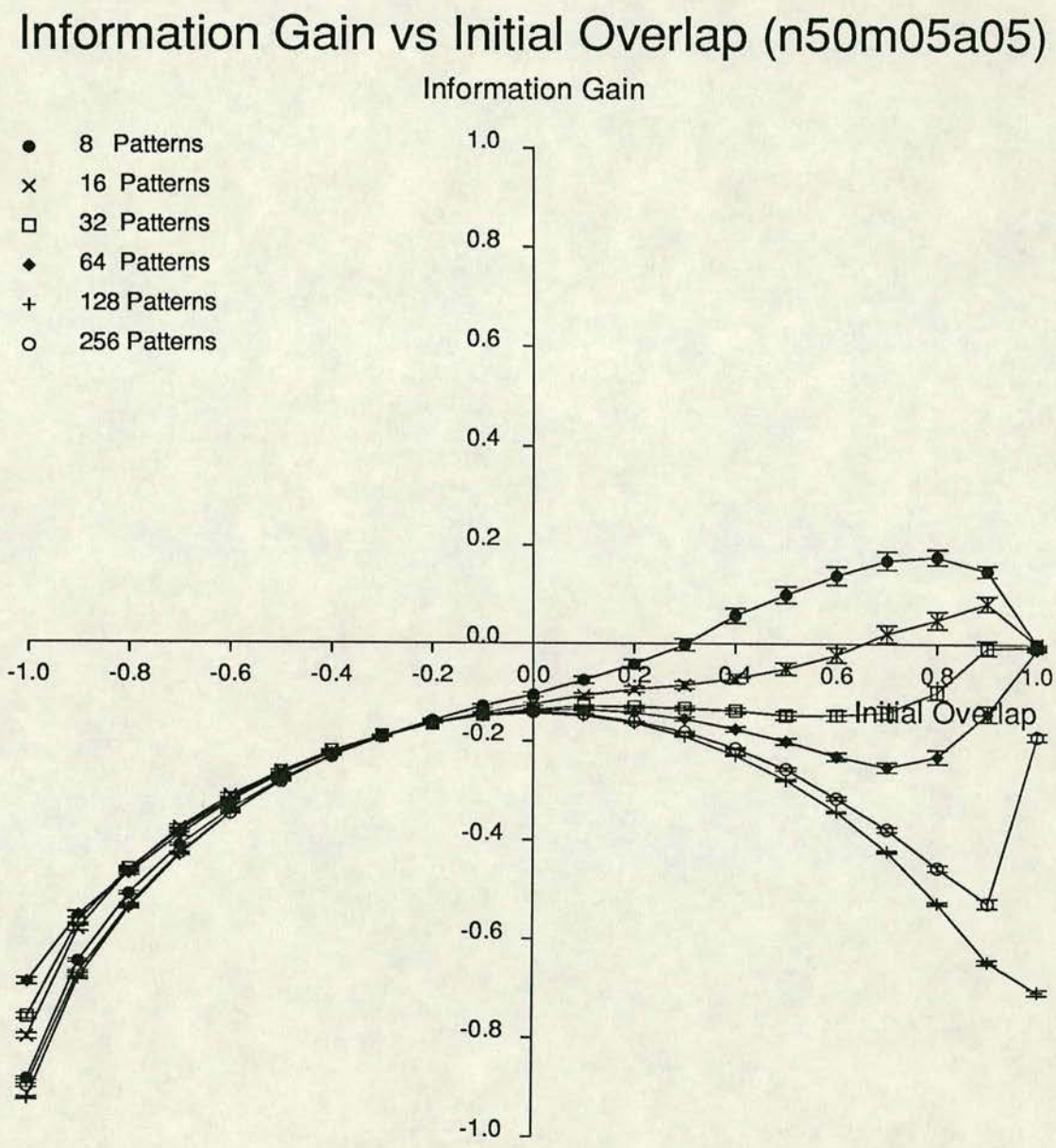

Figure 5.26: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has fifty hidden nodes and is trained on patterns with $a = 0.5$ and $m_p = 0.5$.

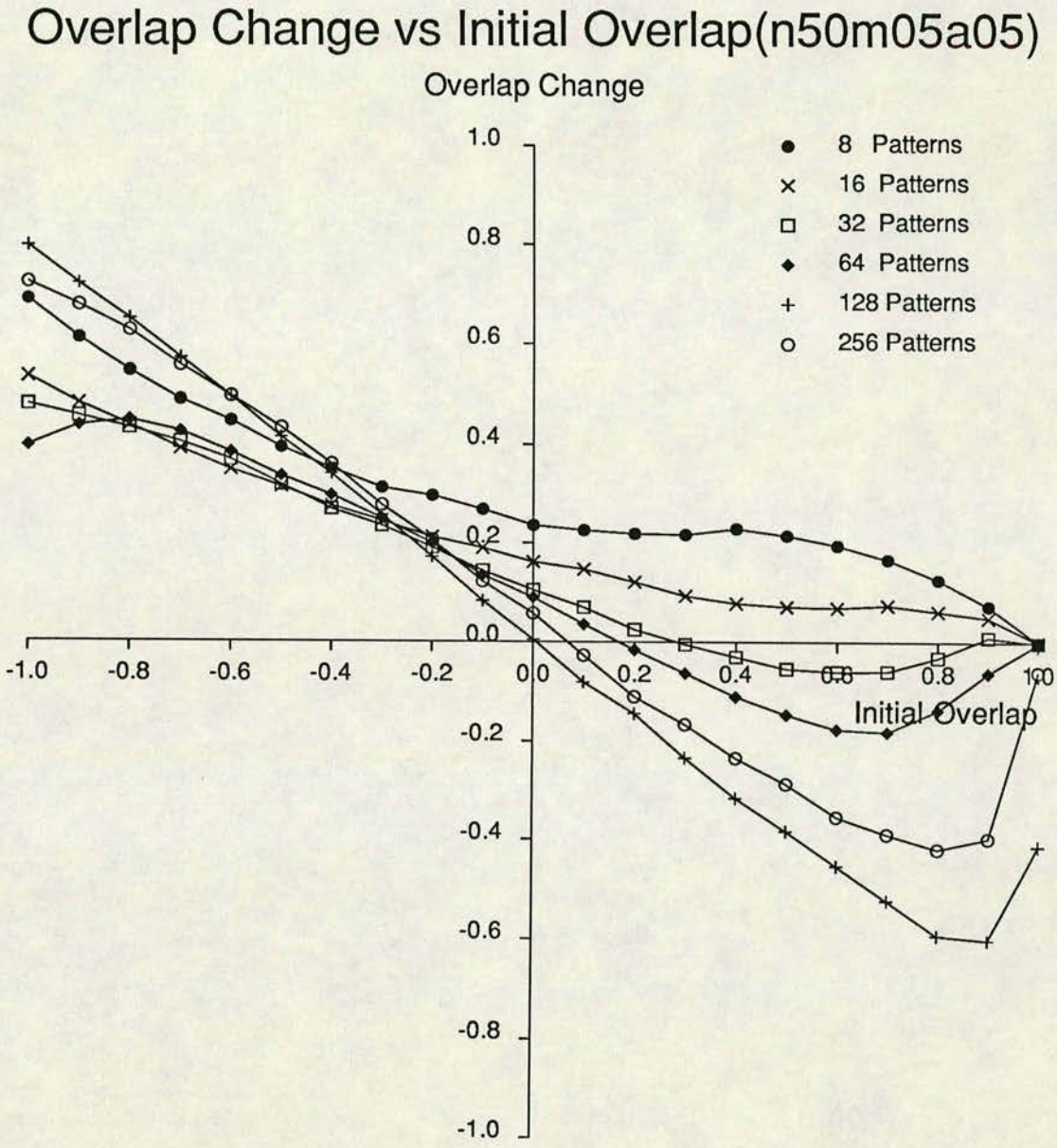

Figure 5.27: Overlap change for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has fifty hidden nodes and is trained on patterns with $a = 0.5$ and $m_p = 0.5$.

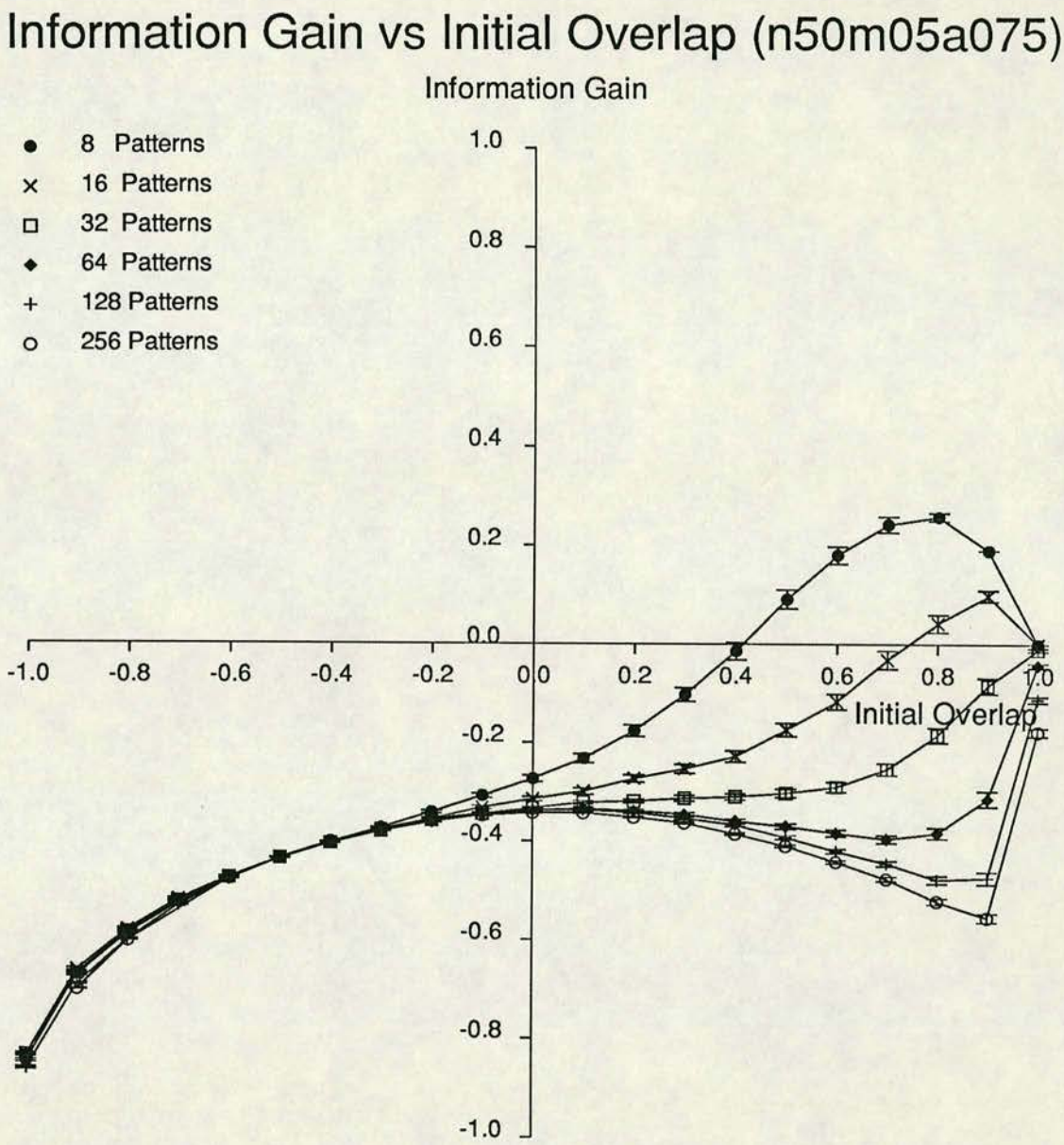

Figure 5.28: Information recovered from noise for various sizes of training set versus the initial overlap between the training patterns and the input patterns. The network has fifty hidden nodes and is trained on patterns with $a = 0.75$ and $m_p = 0.5$.

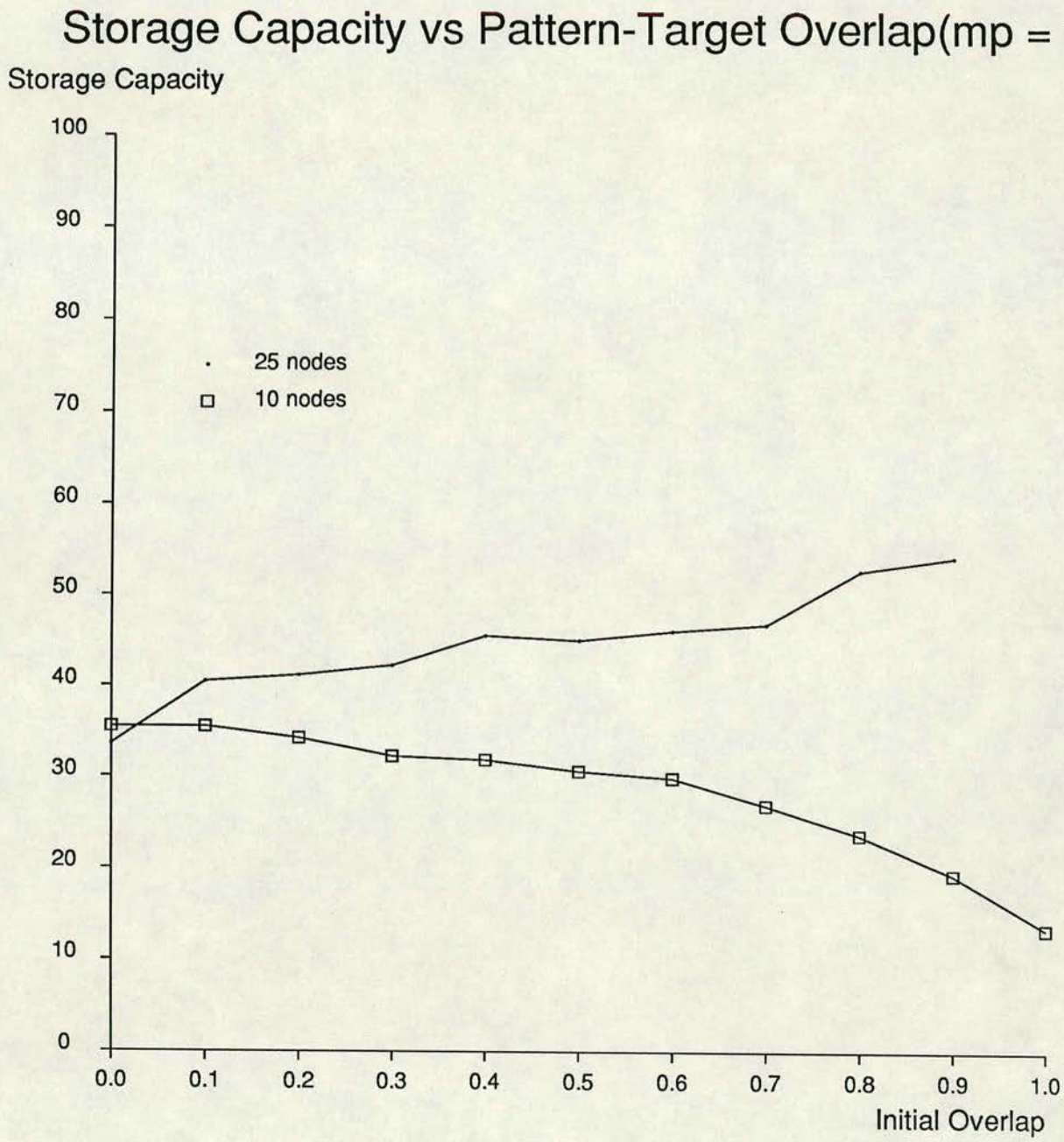

Figure 5.29: The intercept points versus initial overlap for various numbers of hidden nodes with $m_p = 1$.

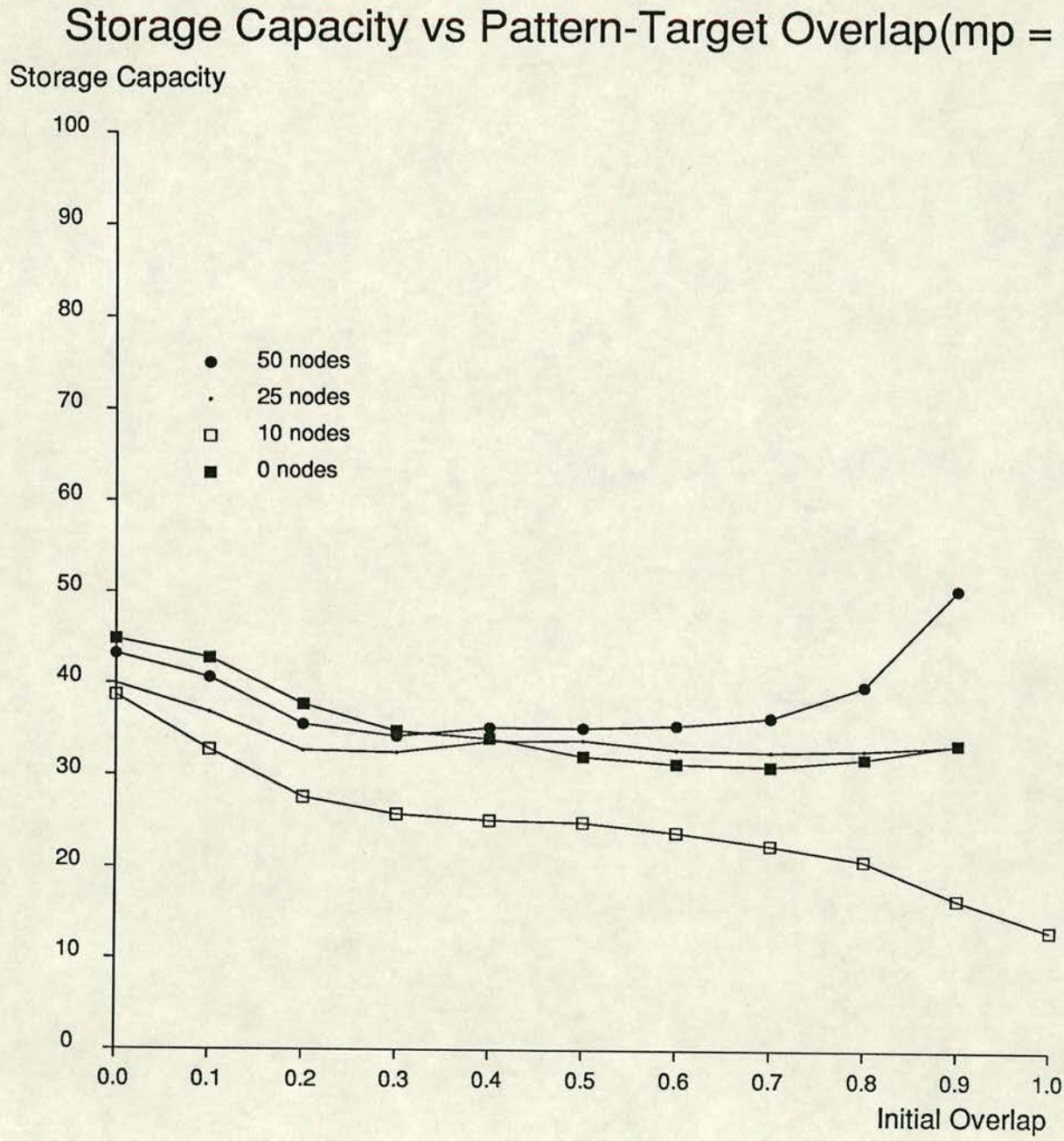

Figure 5.30: The intercept points versus initial overlap for various numbers of hidden nodes with $m_p = 0$.

References

Amit D. J. et al, Phys. Rev. Let **55**, 14 (1985)

Amit D. J., Gutfreund H., Sompolinsky H., Phys. Rev. A, **35**, 2293 (1987)

Bedworth M. D., Bridle J. S., *experiments with the Back Propagation Algorithm: a Systematic look at Small Problems*, RIPRREP/1000/9/87, Research Initiative into Pattern Recognition, RSRE, Malvern, Worcs. (1987)

Bedworth M. D., Bridle J. S., *Using Error Back Propagation: Some Alternatives to Logistic Networks.* SP4 Research Note 75, Pattern Processing and Machine Intelligence Division, RSRE, Malvern, Worcs. (1988)

Beynon T., Dodd N., *The Implementation of Multi-Layer Perceptrons on Transputer Networks*, RIPRREP/1000/13/87, Research Initiative into Pattern Recognition, RSRE, Malvern, Worcs. (1987)

Booth D. M.,*Survey of Statistical Texture Measures*, RIPPREP/1000/37/88, Research Initiative into Pattern Recognition, RSRE, Malvern, Worcs. (1988)

Brodatz P., *Textures: A Photographic Album for Artists and Designers.* Dover, New York, (1966)

Broomhead .D. S. ,Lowe .D. , Complex Systems, **2**, 321-355, (1988)

Canning A., Gardner E., J. Phys. A, **15**, 1657 (1988)

Clarke L. J., *Tiny Version 1.0: Discussion and User Guide*, ECS Project Note ECSP-UG-9, (1989)

Dodd N. *Texture Discrimination Using Multi-Layer Perceptrons*, RIPPREP/1000/15/88, Research Initiative into Pattern Recognition, RSRE, Malvern, Worcs. (1988)

Dodd N. The Rank Prize Funds Mini-Symposium on Neural Network Computation 20th - 23rd February (1989)

ECS, Edinburgh Concurrent Supercomputer Project Directory 2nd edition, Edinburgh Concurrrent Supercomputer Project, University of Edinburgh (1989)

Forrest B. M., J. Phys. A: Math. Gen., **21**, 245 (1988)

Forrest B. M., J. Phys. France, **50**, 2003-2017 (1989)

Gardner E.,Europhys. Lett., **4**, 481 (1987)

Gardner E. J., Derrida B., *Optimal Storage Properties of Neural Network Models*, J. Phys. A: Math. Gen. **21**, 271-284, (1988)

Gardner E., *Optimal Basins of Attraction in Randomly Sparse Neural Network Models*, J. Phys. A: Math. Gen. **22**, 1969 -1974, (1989)

Hebb D. O.,*The Organisation of Behaviour*, Wiley, New York (1949)

van Hemmen J. L., Phys Rev. A **36**, 1959 (1987)

Hinton G. E. and Sejnowski T. J., Proc. 5th Ann. Conf. Cog. Sci. Soc. Rochester, N.Y. p1 May (1983)

Hopfield J. J., Proc. Natl. Acad. Sci. USA **79**, 2554-2558, (1982)

INMOS, *Preliminary Data* IMS T800 it Transputer, April (1987)

INMOS, *Occam2 Reference Manual*, Prentice-Hall 0-13-629312-3 (1988)

Little W. A., Shaw .G .L., Math. Biosci., **39**, 281 (1978)

McCulloch N., Bedworth M., Bridle J., NETSPEAK - *A Multi-Layer Perceptron that can Read Aloud*, RIPRREP/1000/4/87 Research Initiative into Pattern Recognition, RSRE, Malvern, Worcs. (1987)

McCulloch W. S., Pitts W.,*A Logical Calculus of the Ideas Immanent in Neural Nets*, Bull. of Math. Biophysics, **5**, 115-137 (1943)

McCulloch W. S., Pitts W., *How We Know Universals*, Bull. Math. Biophysics, **9**, 127-147 (1947)

Minsky M. L., Papert S. A., *Perceptrons*, MIT Press, (1988)

Nadal J, Toulouse G, *Information Storage in Sparsley Coded Memory Nets*, Submitted to Network: Computation in Neural Nets (1989)

Parker D. B., *Learning Logic*, Invention Report, **S81-64**, File 1, Office of Technology Licencing, Stanford University, (1982)

Parker D. B., To be submitted to the First IEEE Annual Conference on Neural Networks, January (1988)

Prior D. M. N., Radcliffe .N. J., Norman M. G., and Clarke L. J, *What Price Regularity*, Concurrency: Experience & Practise, to appear (1989)

Richards G. D., Proc. 8th Tech. Meeting of the Occam User Group, Sheffield (1988).

Richards G. D. and Tollenaere T., *Documentation for Rhwydwaith Version 2.1*, ECS Note ECSP-UG-7 (1989)

Rosenblatt, F., *Principles of Neurodynamics*, Spartan Books (1962)

Rumelhart D. E. et al D. E. Rumelhart, J. L. McClelland and the PDP research group (editors), *Parallel Distributed Processing: Exploration in the Microstructure of Cognition. Volume 1: Foundations.* MIT Press, Cambridge, MA, (1985).

Rumelhart D. E., Hinton G. E. , Williams R. J., Nature, **323** 533 (1986)

E. Shannon, *A Mathematical Theory of Communications*, Bell Systems Tech. J., **27**, 379-423, (1948a)

E. Shannon, *A Mathematical Theory of Communications*, Bell Systems Tech. J., **27**, 623-625, (1948b)

Smieja F. J., Richards G. D., *Hard Learning the Easy Way: Backpropagation with Deformation* Complex Systems **2**, 6, 671-704, (1988)

Smieja F. J., MLP *Solutions, generalisation and Hidden-unit Representation*, Proc. DANIP Workshop (1989)

Tollenaere T., SUPERSAB*: Fast Adaptive Backprop with Good Scaling Properties.* Submitted to Neural Networks (in press) (1989)

Werbos P. J., *Beyond regression: New tools for prediction and analysis in behavioral science*, Unpublished Ph.D thesis, Harvard University, Cambridge, MA, (1974)

Werbos P. J., *Generalization of backpropagation with application to a recurrent gas market model.* Neural Networks, **1**, 339-356, (1988)

Wiener N., *Cybernetics, or Control and Communications in the Animal and the Machine*, Hermann, Paris; Technology Press MIT, Cambridge, Massachusetts; Wiley, New York,(1948)

Willshaw D. J., Nature **222**, 5197, 960-962, (1969)