ANALYSING AND ENHANCING THE PERFORMANCE OF ASSOCIATIVE MEMORY ARCHITECTURES

SIMON PAUL TURVEY

A thesis submitted in partial fulfilment of the
requirements of the University of Hertfordshire for the
degree of Doctor of Philosophy

The programme of research was carried out in the Department of Computer Science,
Faculty of Engineering and Information Sciences, University of Hertfordshire

February, 2003

Abstract

This thesis investigates the way in which information about the structure of a set of training data with 'natural' characteristics may be used to positively influence the design of associative memory neural network models of the Hopfield type. This is done with a view to reducing the level of connectivity in models of this type.

There are three strands to this work. Firstly, an empirical evaluation of the implementation of existing theory is given. Secondly, a number of existing theories are combined to produce novel network models and training regimes. Thirdly, new strategies for constructing and training associative memories based on knowledge of the structure of the training data are proposed.

The first conclusion of this work is that, under certain circumstances, performance benefits may be gained by establishing the connectivity in a non-random fashion, guided by the knowledge gained from the structure of the training data. These performance improvements exist in relation to networks in which sparse connectivity is established in a purely random manner. This dilution occurs prior to the training of the network.

Secondly, it is verified that, as predicted by existing theory, targeted post-training dilution of network connectivity provides greater performance when compared with networks in which connections are removed at random.

Finally, an existing tool for the analysis of the attractor performance of neural networks of this type has been modified and improved. Furthermore, a novel, comprehensive performance analysis tool is proposed.

Acknowledgment

It goes without saying that a project of this magnitude enormously affects one's own life and indirectly touches so many others. There are, therefore, a correspondingly large number of people to thank.

My supervisory team must be singled out for special praise:

Steve Hunt, for his unwavering faith, sublime patience, and occasional yet splendid hospitality.

Neil Davey, for his bubbling enthusiasm, constant good humour, and continuous, yet gentle, encouragement.

Ray Frank, for acting as a foil to Steve and Neil, underscoring and enhancing their unique and distinctive characteristics.

I must thank my parents, especially my father, for bringing me up and shaping me into the person I am today. My office-mates, researchers all, whose hard work has inspired me to stay true to my own task. Without their laughter, conversation, discussion, and, importantly, coffee breaks, none of this would have been possible.

Around the university, so many others have lent their time in so many different ways: Lynette, Bob, Austen, the department's computer technicians. All these people and more have, in no small measure, contributed indirectly to this work.

Finally, much closer to home, I must thank my partner Cat for her continuing love and patience. She and our various companions, George W. Bear, Eeyore Belle, Magic, Scorch, and Mrs Scorch, have managed to keep my spirits high and a smile on my face. I could not have done it without you. Also, my good friends Alex and Katherine; their continuous friendship and support was both welcome and appreciated.

Thank you all.

Table of Contents

List of Tables

List of Figures

1. INTRODUCTION

1.1. Introduction

The purpose of this thesis is to answer the question: "*What benefits are there to using information about the task in guiding the design of the pattern of connectivity of a sparsely connected Hopfield-type neural network?*".

There are three themes to this work: Firstly, the empirical investigation of existing theory; secondly, the combining of theories to produce novel network models and training regimes; thirdly, the proposal of new strategies for constructing and training associative memories.

Hopfield-type networks used for research purposes are usually trained on random bit-patterns and the structure and correlations present in more natural data are not taken into account. Little work on relating sparse connectivity to the nature of the training data exists and even less is available empirically evaluating any resulting network models. Therefore, this thesis is not a theoretical work but rather an empirical evaluation of a number of architectural modifications to the original Hopfield network. The modifications made however will be shown to be based on established, published theory.

The key results are presented in chapters 8 and 9. It is demonstrated that, under certain conditions, correlations in the training data can be exploited through particular patterns of connectivity and that this can lead to improved capacity and attractor performance.

Finally, while biological plausibility is not a driving factor in this investigation, implausibility is avoided wherever practicable. The results obtained therefore, may well be interesting from both an engineering and biological standpoint.

1.2. Methodology and Research Goals

The methodology employed throughout this work is to conduct empirical evaluations based on many averaged network simulation runs using both random and designed datasets.

A variety of distinct tasks needed to be completed in order to accomplish the overall aim of the investigation. These were:

1. To investigate the current state of the art with respect to Hopfield-type associative memories. High performance learning rules and performance metrics were of particular interest. A learning rule was to be chosen for use in later experiments.
2. To develop a neural network simulator suitable for immediate use and capable of being extended for later, further experimentation.
3. To manufacture sets of training data that simulate the structure of natural patterns and to investigate the nature of intra- and inter-pattern correlation in the manufactured data sets
4. To investigate the history and current standing of the field of sparsely connected associative memory architectures.
5. To investigate the impact on network performance of post-training removal of connectivity. As a simple method of reducing network connectivity, this strategy needs to be investigated for purposes of later comparison.
6. To investigate the effectiveness of two techniques for constructing sparse connectivity prior to training. The first of these will be a simple random connectivity strategy; the second will create connectivity based on some knowledge of the structure of the training data.
7. To investigate whether the attractor performance of sparsely connected networks can be improved with additional connectivity.

1.3. Thesis Outline

The structure of the thesis is as follows:

Chapter 2 presents a brief history of the Hopfield network together with an explanation of the architecture and the dynamics of the network. A review of the background literature pertinent to this investigation is included as is a presentation of a number of different learning rules applicable to the basic Hopfield architecture. An explanation of a method of categorisation of the weight matrices resulting from the presented learning rules is given.

Chapter 3 describes a number of measures used in assessing the performance of the networks created in the course of this investigation. Issues with existing tools are identified and solutions to them are proposed. Also presented is a new attractor performance measure providing the same functionality as existing tools while extending and improving the quality and quantity of analytical information provided.

Chapter 4 presents the results of the application of the performance tools described in chapter 3 to fully-connected networks trained using the learning rules described in chapter 2. The learning rules are evaluated according to their performance and a learning rule is chosen for use in further work.

Chapter 5 introduces the field of sparsely connected associative memory networks. A justification of the approach taken in this investigation to establishing sparse connectivity is provided based on a number of existing works. Various techniques used to establish sparse connectivity are described. A review is then presented of the existing literature related to sparse connectivity in the context of associative memories.

Chapter 6 presents the results of a series of experiments using networks in which connectivity has been removed after training. Training is carried out using two learning rules. The first of these is the learning rule identified for future use in chapter 4; the second is another high-performance learning rule from chapter 2, included for purposes of comparison. The performance of the networks is evaluated with respect to the level of pattern stability and attractor performance at each level of connectivity.

Chapter 7 presents the results of analyses performed on non-random training data carried out in preparation for future work. The pseudo-natural datasets are introduced and the characteristics of such data explained. The analysis tools are described alongside details of their use. Finally, the results of applying the analysis tools are presented and the implications of the results discussed.

Chapter 8 deals with the subject of creating Hopfield-type associative memories with structured sparse connectivity and two methods of creating sparse connectivity are described and justified. Networks are trained with various numbers of input patterns and results of five types of performance analysis are presented. The implications of the results are then discussed.

Chapter 9 builds on the results of chapter 8 and presents the results of an investigation into ways the attractor performance of the networks might be further improved once an initial level of connectivity has been established locally. Firstly, a technique for correcting networks which exhibit some degree of error in their training is investigated. Secondly, the effect of adding further connectivity to the networks is examined. The networks are assessed with respect to the attractor performance metric described in chapter 3.

Chapter 10 concludes the thesis and summarises the findings described in the earlier chapters. The novel areas of work are identified and some practical implications of the work discussed. Finally, some potential avenues for future work are identified.

1.4. Thesis Format

The chapters of this thesis are numbered sequentially from 1 and are identified in the text by being preceded by the word 'chapter'. Sections exist within chapters and are identified along with the chapter number to which they belong. Cross-references to sections begin with the section symbol §, such that §3.1 would refer to the first section in the third chapter. Figures and tables are similarly labelled. For example, 'figure 4.2' refers to the second figure in the fourth chapter. Tables are denoted by using the word 'table' in place of 'figure' where appropriate. Figures and tables are indexed separately. In practice, this means that both 'figure 2.1' and 'table 2.1' could exist in chapter 2, the former referring to the first figure and the latter, the first table.

1.5. Common Notation

For convenience, some frequently used notation is identified and defined.

The letter N is frequently used to represent the number of neurons in a network.

Neuron indices are usually represented using the letters i and j.

A network's current state vector is represented by the letter **S** and the state of an individual neuron represented by S_i.

An individual training vector is represented by the letter $\boldsymbol{\xi}$ and an individual bit represented by ξ_i.

A network's weight matrix is denoted using the character **W** and individual weight are referred to using the notation W_{ij}. This represents the weight on the connection between from neuron j to neuron i.

The local field of an individual neuron, i, is denoted h_i

2. A SURVEY OF THE HOPFIELD AND RELATED NETWORK MODELS

2.1. Overview

It is arguable that the entire modern field of associative memory neural networks stems from the seminal work of Hopfield (1982; 1984) in which the physical properties of the spin glass provide inspiration for a neural network. Kinzel (1987) provides an introduction to the relationship between spin glasses and neural networks. In that work spin glasses are described as being disordered magnetic materials in which, at low temperatures, the atoms freeze to form a random structure. The analogy is formed between the atoms of the spin glass as neurons, and the magnetic forces through which they interact as synapses.

With Hopfield's publications came a resurgent interest in recurrent networks and content addressable memories and whilst undeniably the catalyst for this renaissance, Hopfield was neither the first nor alone in investigating spin glass-like models. For instance, Little (1974) suggested a model bearing similarities to that proposed by Hopfield. As observed by Gurney (1997) however, the quantum mechanical slant to Little's work may have prevented it from being the genesis of the modern field that Hopfield's paper was to become.

Even earlier than Little's work, Willshaw (1969) proposed a simple associative network based on the principles and properties of the hologram. This work bears little resemblance to the Hopfield model as it is known today but nevertheless incorporates thresholded neurons at its core and so registers as a somewhat distant cousin. Gardner-Medwin (1976) examined the similarities with respect to recall and update dynamics between the brain and recurrent networks mainly of the Hebb (1949) type.

2.2. Hopfield Networks

Hopfield's original network (1982) uses binary or bipolar neurons derived from the work of McCulloch and Pitts (1943). A neuron is always in some state. In the case of binary neurons the state takes the form of one of the values 0 and 1 and for bipolar neurons the state is represented by the values -1 and +1. The current state of neuron i is denoted S_i where the symbol i indicates the index of the neuron and takes the value 1..N, where N is the number of neurons in the network. The current state of the network as a whole is represented by the *state vector* **S**.

Each neuron has an activation threshold against which its input is evaluated. The decision as to whether or not an individual neuron should fire is wholly based upon the value of the neuron's input relative to the activation threshold. The network is fully-connected in that each neuron is connected to all others. This pattern of interconnection makes networks of this type recurrent in nature, as connections feed information back to other neurons.

The connections between neurons (often termed synapses) are bidirectional and have values associated with them known as weights. The matrix of values representing these weights is called the *weight matrix* and is given the symbol **W**.

An individual connection links the output of one neuron to the input of another and the weight value can affect the traversal of the connection by some signal emitted by the outputting neuron. The process of calculating a neuron's input, the determination whether or not it should fire, the value of the signal emitted, and the manner of its propagation to the inputs of other neurons is determined by a network's *update dynamics*.

2.3. Network Dynamics

The dynamics of a recurrent network influence its recall characteristics and performance. Given some start state, the weights and dynamics of a network determine the states through which the network will pass. For networks that possess less than full connectivity, the effect of the pattern of connectivity of the network on the network dynamics cannot be ignored and rather than specific allowance having to be made for such an architecture the absence of a connection can be considered as equivalent to the weight on that connection being zero, as Hopfield suggests. Such a connection will thus have no effect on the dynamics of the network as will be seen below.

2.3.1. Simple Update Dynamics

The original Hopfield network is presented as possessing neurons that randomly and asynchronously evaluate themselves in parallel with regard to whether or not their level of activation, called the *net input* and termed h_i at neuron i, is above or below some threshold value denoted θ.

The calculation of the net input of a neuron is performed as follows:

$$h_i = \sum_{j=1, i \neq j}^{N} W_{ij} S_j \tag{2.1}$$

The next state S_i' of neuron i is calculated from the net input:

$$S_i' = \begin{cases} +1 & \text{if} \quad h_i > \theta \\ S_i & \text{if} \quad h_i = \theta \\ -1 & \text{if} \quad h_i < \theta \end{cases} \tag{2.2}$$

where θ is normally taken as 0.

Equation (2.2) represents the neurons' *output function* and when graphed, excepting the special case where h_i is equal to zero, looks as follows:

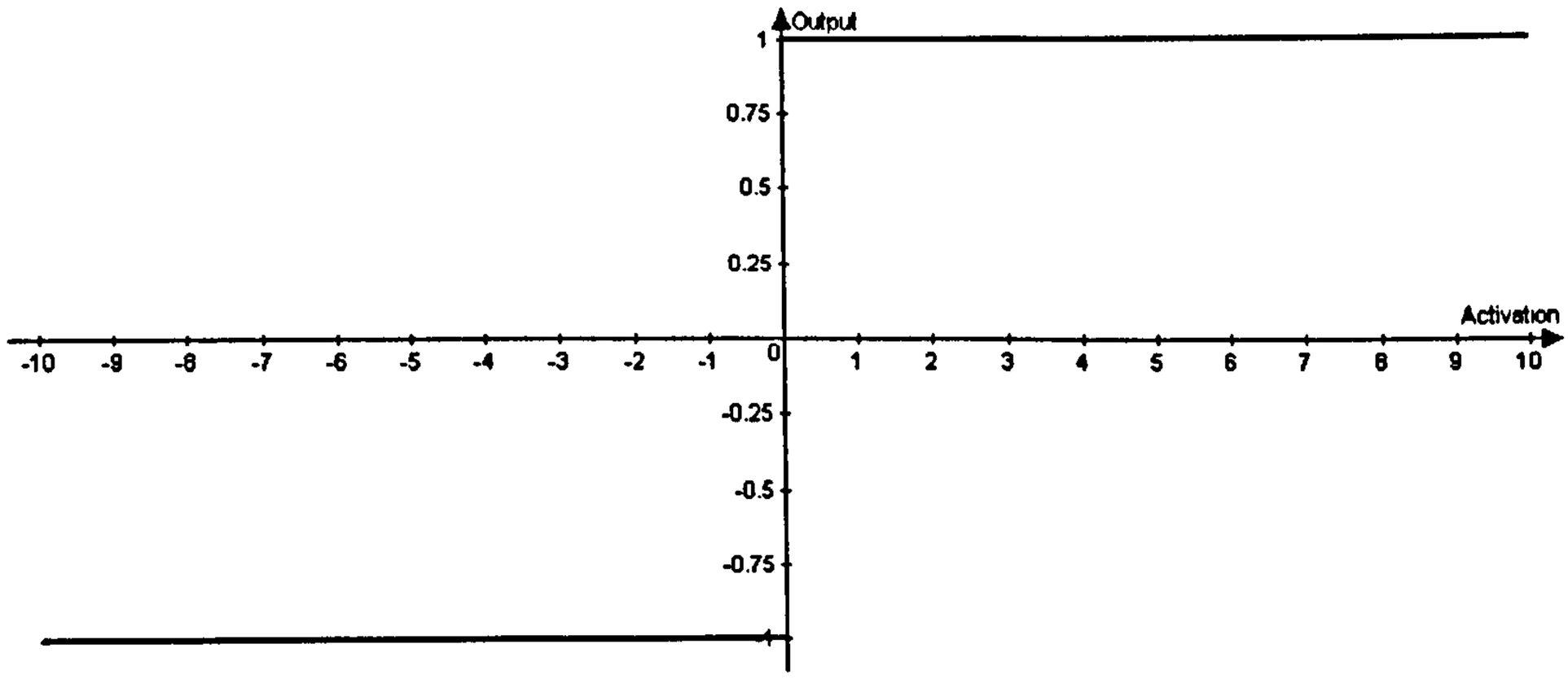


Figure 2.1: Plot of the bipolar step function.

Since true parallelisation is not possible using computer software simulations on non-parallel hardware, sequential equivalents must be employed. Some methods by which this can be accomplished are detailed below.

There are two fundamental methods of updating neurons: synchronous and asynchronous. During synchronous update, all the neurons of a network are updated at the same time while asynchronous update differs in that neurons are updated one by one.

Regardless of whether the update dynamics are synchronous or asynchronous, symmetric weights ($W_{ij} = W_{ji}$) are required for a valid energy function. The presence of an energy function implies simple dynamics in the form of fixed points or *n*-cycles. As the network state evolves according to the rules of the dynamics (equations (2.1) and (2.2)) the energy function never increases. The implication of this is that the stored patterns form local minima of the energy surface described by the combination of equation (2.3) and all 2^N possible states of the network. The energy function for the standard Hopfield dynamics is as follows (Hopfield, 1982):

$$E = -\frac{1}{2}\sum_{i=1}^{N} S_i h_i \tag{2.3}$$

Synchronous Updates

The term synchronous update is somewhat misleading in the context of software simulations of neural networks. It is not possible to genuinely update all neurons simultaneously so an equivalent system must be sought. In synchronous updating, the neurons' outputs are calculated *en masse* before they are fed back to act as inputs to those neurons during the next time step.

Using this method, all neurons are effectively updated simultaneously. This can cause the network to fall into 2-cycles with the network's neurons collectively switching between 2 distinct states; these states are always some pattern and its inverse.

There are a number of problems with the idea of synchronous updating. The notion that all neurons in the brain might update at the same time is clearly flawed. The requirement for some centralised clock with which to synchronise the updates illustrates this. Also, when updating synchronously, there is no opportunity for any neuron to affect the update of any other. It is this fact can lead to a network simply flipping between states and not converging on a solution.

Asynchronous Updates

During asynchronous update, the neurons are selected for update one at a time, in either a fixed or random order. It is the order in which the neurons are selected that is the primary means of distinguishing between the three different ways in which asynchronous updating may be implemented.

During *fixed order* asynchronous update, neurons are simply selected in some fixed order and the output of each one is calculated. The output is immediately available as an input value to neurons yet to be updated. The act of updating all the neurons in the network once is termed an *update iteration.*

Random order, asynchronous updates guarantee convergence on a stable pattern given a symmetric, zero-diagonal weight matrix (Hopfield, 1982). Asymmetric weights may result in the network being unable to converge upon a single state. When this occurs, the network commonly ends up either cycling around a number of states without settling or even wandering chaotically.

Two variations of random order asynchronous dynamics exist. The first of these represents the closest serial analogy to the parallel dynamics proposed by Hopfield. Neurons are randomly chosen for update at discrete time steps. At each time step, each neuron has an equal chance of being selected and over a sufficiently long period of time all neurons should have been updated approximately the same number of times. This variant is termed *random update with replacement* since after update a neuron is replaced in the pool of those available to be selected for update. It should be apparent that the concept of an update iteration cannot apply to this update method.

The second random order variant reintroduces the update iteration and specifies that each neuron may only be selected for update once in each update iteration. In all other aspects this update method operates in the same way as random update with replacement. This variant is termed simply *random update.*

Throughout this work, asynchronous *pseudo-random* updates are employed. For speed, a large table of random values is precalculated and this table determines the order in which neurons are updated.

2.3.2. Stochastic Dynamics

Stochastic neurons are those where the value of the net input to the neuron determines not whether the neuron will fire but rather, the probability that the neuron will fire. The degree of stochasticity in the network is commonly controlled by some notion of 'temperature' that may be gradually reduced as network update progresses.

This idea of the 'temperature' of a system has its roots in the field of thermodynamics where the temperature of an entity is related to the level of energy in the system. When applied to neural networks the pseudo-temperature regulates the amount of 'random' movement in the network. At a temperature of 0 a network becomes deterministic in nature. These dynamics can be seen in alternative associative memory models such as the Boltzmann machine (Hinton and Sejnowski, 1983).

The purpose behind this introduction of noise (the random movement) is to prevent the system falling into spurious local energy minima and to aid its movement into one of the energy wells of one of the stored patterns.

A stochastic version of the Hopfield network exists (Hopfield, 1982) whereby the convergence state of the network is measured as the average state of each neuron over some period of time. In this model, the temperature is not reduced and so the network constantly updates with some degree of randomness.

2.3.3. Continuous Dynamics

In introducing the network update dynamics it was shown, using equation (2.2), that the Hopfield network is often constructed using a step output function. Other output functions are possible and are applicable to the Hopfield architecture.

Continuous Hopfield Network

Hopfield (1984) proposes the construction of a model based on a sigmoidal output function. Hopfield shows that, under certain conditions, the stable states of the continuous model correspond to those present in the equivalent discrete network as, over time, the network saturates out to +1/-1 states. The justification of this output function lies in the fact that it may be viewed as representing the short-term average of a biological neuron's firing rate.

Morita Dynamics

Morita (1993) presents an interesting modification to the update rule. Whereas Hopfield-style associative memories are commonly built using a sigmoid output function (continuous Hopfield model) or a step function (discrete Hopfield model). Morita presents a non-monotonic output function that, it is claimed, provides better performance.

The output function is given by:

$$f(u) = \frac{1-\exp[-cu]}{1+\exp[-cu]} \cdot \frac{1+\kappa \exp\left[c'(|u|-h)\right]}{1+\exp\left[c'(|u|-h)\right]} \tag{2.4}$$

Where, in this case, c, c', and h are positive constants and κ is a parameter which is usually negative. A graph of this function is shown below (figure 2.2).

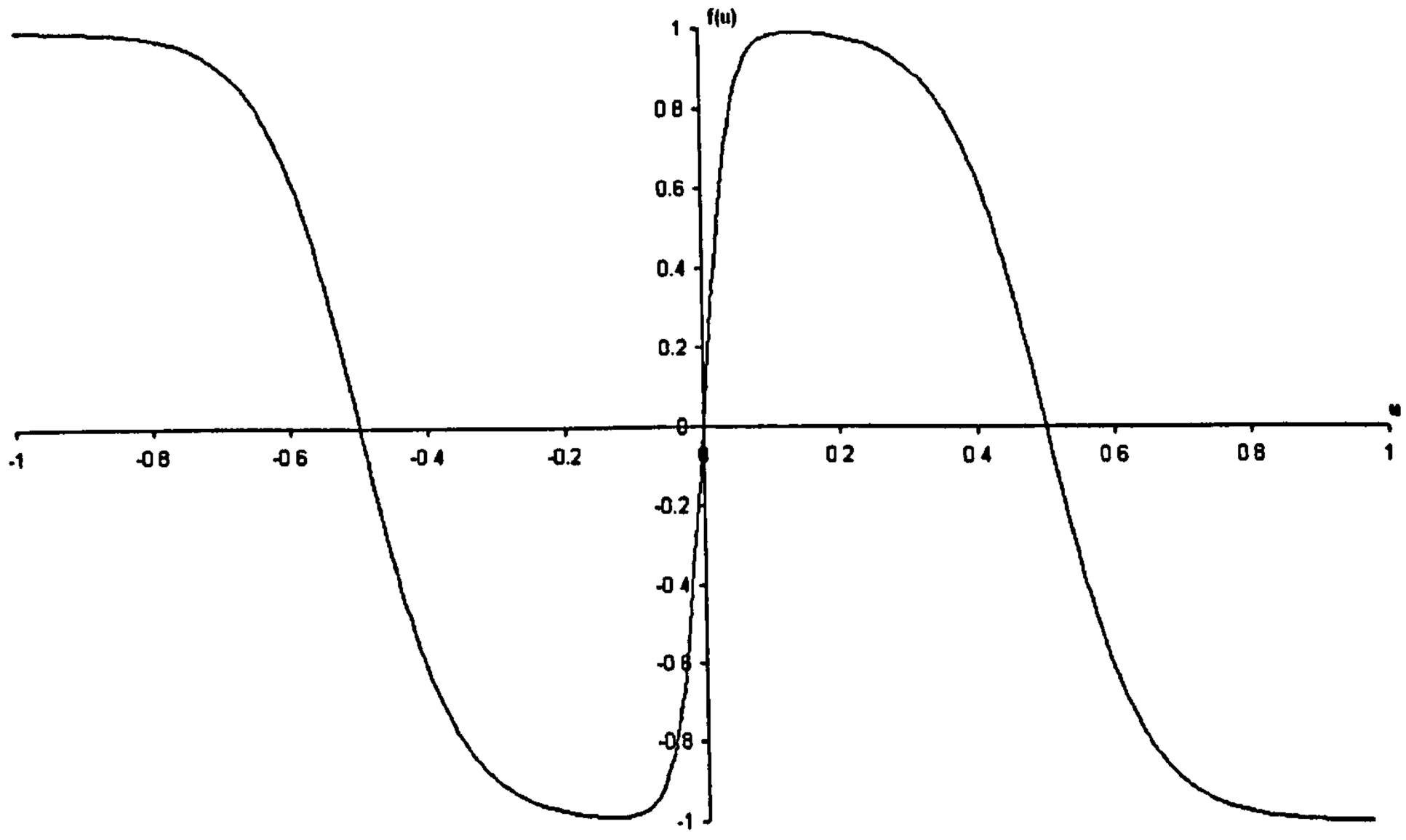


Figure 2.2: Output function employed by non-monotonic Morita dynamics. The parameters used to generate the graph were: c=50, c'=15, h=0.5, and κ=-1.

Morita states that the use of this output function in both continuous and discrete networks greatly improves recollection ability and memory capacity. Morita also notes that the continuous model no longer recalls spurious memories with the modified dynamics.

Whilst the claimed attributes of Morita's update rule are undoubtedly of interest, a detailed investigation of its implementation is outside the immediate scope of this work.

2.4. Weight Matrices

The values possessed by the weight matrix of a particular instance of a Hopfield-type network is determined in part by the algorithm employed in embedding the patterns which are to be learnt and in part by the patterns themselves. The algorithms used to train networks to recognize these patterns are commonly termed *learning rules.*

The learning rule used to train the original model was inspired by Hebb (1949). The assumption is that persistent or repetitive activity at a neural level induces lasting (though not necessarily permanent) changes that add to the increased embeddedness of the associated pattern. Hebb states this as follows:

> *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or more cells such that A's efficiency, as one of the cells firing B is increased."*

In its original form above the Hebb proposal is not explicit enough to form part of a working model. Quoting from Haykin (Haykin, 1999), the simplest interpretation (Stent, 1973; Changeux and Danchin, 1976) of the Hebb proposal into a workable training prescription is:

> *"If two neurons on either side of a synapse (connection) are activated simultaneously (i.e. synchronously), then the strength of that synapse is selectively increased."*

This method will produce a working weight matrix though it fails to make use of all the information available to it. In the prescription outlined above there is no distinction between correlation between neurons that are off together (0/0), and neurons which are anticorrelated (0/1) in the case of a binary representation.

The first interpretation of Hebb's proposal then becomes deficient for the purpose of generating a suitable weight matrix. The natural evolution of the interpretation given above is to take into account the positive and negative activation values present in a bipolar representation. This modified scheme is easily presented as a system of neural learning based upon the principles of correlation/anti-correlation. If two units of a network are in agreement with each other, i.e. are both outputting +1 or –1, then the synaptic strength between them, represented in this case by the

weight on the connection joining the units, should be increased by some amount. If, on the other hand, the two units in question are outputting different values from each other then they are in disagreement and the weight should be correspondingly decreased. As an equation for forming a weight matrix for the network the correlation/anti-correlation method of learning can be represented thus:

$$W_{ij} = \frac{1}{N}\sum_{\mu=1}^{P} \xi_i^{\mu}\xi_j^{\mu} \qquad W_{ii} = 0 \tag{2.5}$$

where W_{ij} represents the weight on the connection from neuron j to neuron i and μ represents the index of the current training pattern. Summing the correlations over all patterns for each neuron pair ij, gives the value for the weight between that pair.

The condition $W_{ii} = 0$ enforces the zero diagonal required for guaranteed convergence using the asynchronous random network update dynamics described in §2.3.

By training the network in this manner we are reinforcing correct performance and 'punishing' (through reduction of the weights) the incorrect behaviour.

The Hopfield model based on Hebbian learning has a relatively low capacity when compared with models trained using later algorithms. However, it does have the distinction of possessing three important properties that cause it to be generally accepted as being a plausible, though unlikely, model of biological neural interaction. These three properties (denoted by bold type) are detailed below:

- The algorithm is **local** in its use of information. Two forms of locality are identified: *spatial locality* and *temporal locality*.

  If an algorithm is spatially local, the information that the algorithm requires in order to calculate the weight change on the synaptic connection is that which is directly available to the two units between which the connection exists.

  For an algorithm to be temporally local, the information that the algorithm uses must exist wholly and exclusively at the current point in time and comprise only that information contained within the network's state and weights.

- The algorithm is **immediate** in its effect. That is, the algorithm requires only a single pass through the training set in order to calculate an effective weight matrix. A learning rule operating in this way is often called a *one-shot* learning

rule. Other forms of learning rule can require a number of passes through the training set before a suitable weight matrix is formed and these are termed *iterative* learning rules.

- The algorithm is **incremental** in the way in which it learns new patterns. If a learning rule is described as being incremental it is taken to mean that the algorithm is capable of embedding new memories in the network without reference to, or destroying, those that already exist.

There exist minor variations on equation (2.5) as shown by Müller and Reinhardt (1991). They suggest that for a network that excludes the self-coupling of units a normalisation factor of $1/(N-1)$ would be appropriate as the summation runs over $N-1$ terms.

As alluded to in §2.1 the Hebb rule is not the only means by which networks of the Hopfield type may be trained.

2.4.1. Abbott's Network Classes

According to Abbott (1990) it has been shown (Abbot and Kepler, 1989) that associative memories fall into *universality classes* identified by the networks' behaviour near maximum capacity. Networks may exhibit quite different behaviour from each other at lower loadings but those belonging to the same class begin to behave similarly as they reach saturation. The importance of these universality classes lies in the fact that it becomes less important that the model being studied is an absolutely accurate representation of the real system. If the model being studied lies in the same class as the real system it is derived from then calculations upon the model will elicit the same answers as if they were performed on the real system.

The universality classes are defined by the distribution of *stability coefficients* known as *gammas*. At first glance, it could seem obvious that the size of the local field, h_i, might be sufficient to indicate the embeddedness of a particular pattern at unit *i*. While it is true that, according to the combination of equations (2.1) and (2.2), it is enough for the value $h_i\xi_i$, termed the *aligned local field*, to be positive to ensure pattern stability; this would imply that simply multiplying all the weights of a network would improve the attractor performance of the patterns. This is not the case however, as it is not the magnitude of the weights that is important for pattern stability but rather the size of the weights relative to one another. Scaling every weight by the same value will thus have no effect on the stability or otherwise of the patterns in the training set.

Removing the aligned local field's dependence on the magnitude of the weights is the key to eliminating this potentially misleading feature.

Stability coefficients, as their name implies, provide an indication of the depth of embedding of a trained pattern and are calculated as follows:

$$\gamma_i^\mu = \frac{h_i^\mu \xi_i^\mu}{|W_i|} \tag{2.6}$$

where:

$$|W_i| = \left(\sum_{j=1}^{N} W_{ij}^2 \right)^{1/2} \tag{2.7}$$

The denominator $|W_i|$ present in equation (2.6) makes the calculation of gamma independent of any scaling of the weights. Now, if the gamma values for a pattern are all greater than zero then it can be said that that pattern will be a stable point of the network.

In the simplest case, a network consisting of random weights would generate a distribution of gammas similar to the following:

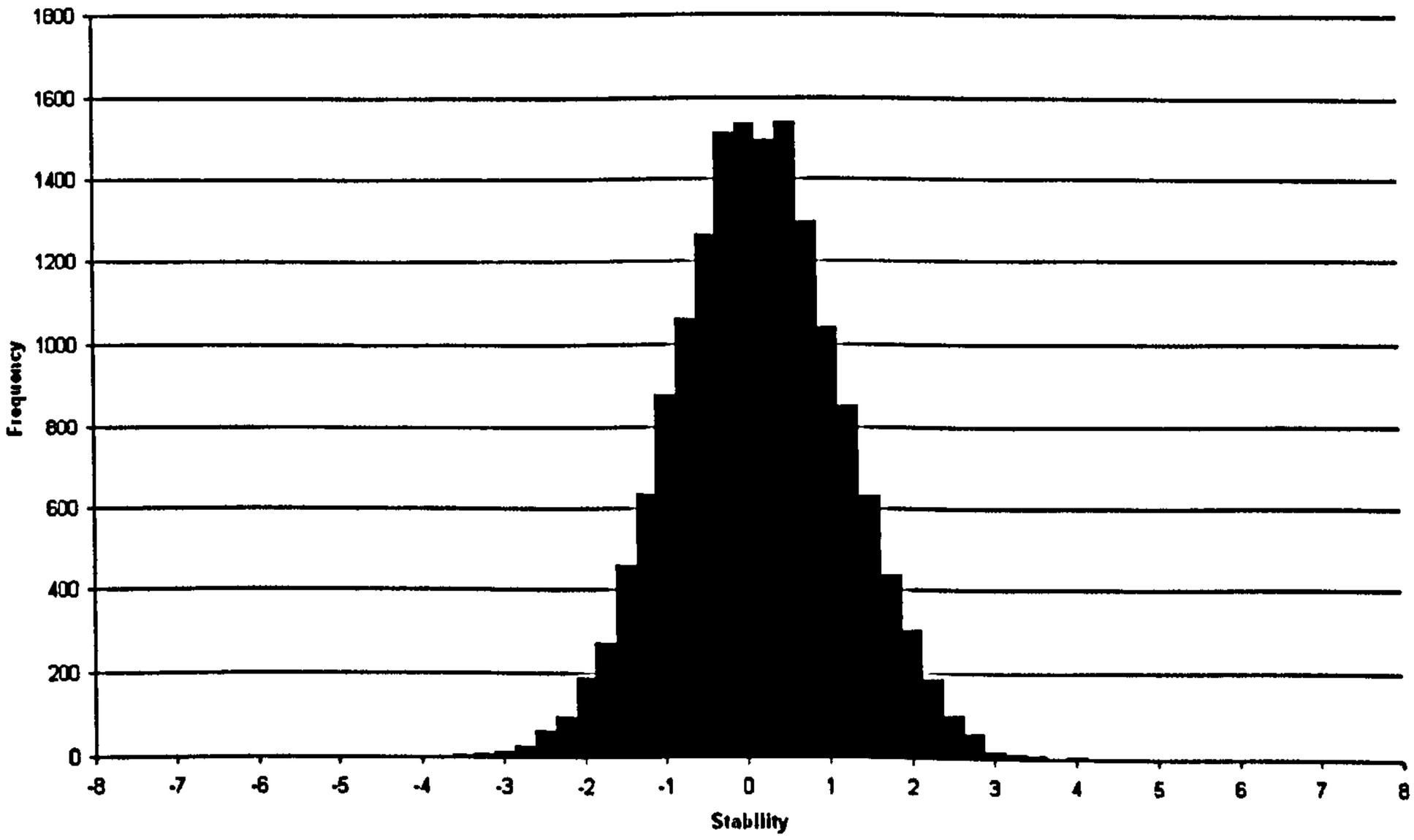


Figure 2.3: Gamma distribution for a network with random weights.

As might be expected from a matrix of random weights, the distribution of gamma values is approximately Gaussian with a zero mean. It can be seen that roughly half the values fall below the stability threshold of zero. This worst-case class of weight matrices will be termed *Class 0*.

Class 1

In order to improve the stability of the training set there are two immediate strategies that might be employed. They are: shifting the mean of the distribution so that a greater number of values become greater than zero, and tightening the distribution which will have a similar effect in pulling all the values closer together.

Weight matrices exhibiting these properties can be generated using Hebb-like learning rules. Figure 2.4 (below) shows an example of the distribution of gammas one might expect from such a matrix.

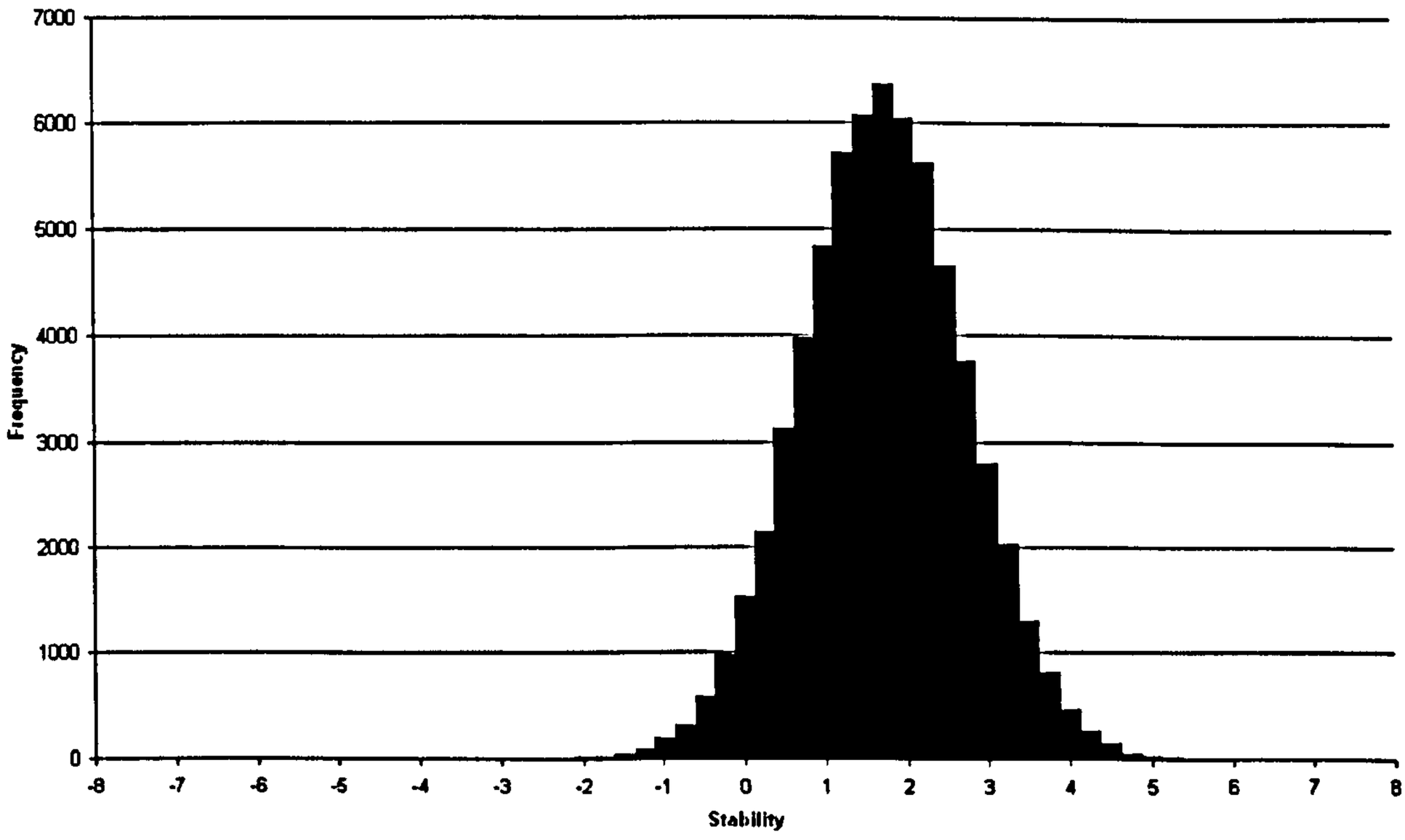


Figure 2.4: Gamma distribution for a network with weights generated using the Hopfield learning rule.

The distribution shown in figure 2.4 has a mean of approximately 2 and a smaller proportion of values below the stability threshold. The weight matrix that the distribution was produced from was generated by the Hopfield learning rule. Weight matrices that have distributions of gamma values similar to that above are said to be *Class 1* matrices. This category is also known as the *Hopfield class* as it is the distribution of gammas from the Hopfield weight matrix that is the archetype of this class.

Networks with weight matrices of this type commonly have very low capacities of around α=0.15. Abbott's analysis of the Hopfield architecture shows however, that a matrix should exist with a narrower Gaussian distribution of gammas with α=1.14. Abbott neither specifies nor suggests a technique by which such a matrix might be generated.

With class 1 rules, there is always a non-zero probability that a trained pattern may not be stable.

Learning rules that generate weight matrices of this class are Hebbian learning (Hebb, 1949; Hopfield, 1982; Hopfield, 1984) as described above and the rule developed by Storkey (1997; 1999).

Class 2

In figure 2.4, the presence of values below the stability threshold indicates that not every bit in every pattern is stable for the network from which that distribution was obtained. The production of another class of weight matrices is possible by learning rules which rectify this situation by moving the distribution mean to some positive value (1, in the simplest case) and attempting to make the variance of the data zero.

This class of weight matrices is known as *Class 2* or the *pseudo-inverse class* and is so called because the algorithms that fall into this category all calculate or approximate the weight matrix based on the calculation of the pseudo-inverse of the matrix formed by the training patterns. According to Amit (1989), this technique was originally suggested by Kohonen *et al* (1973) and adapted for application to neural networks by Personnaz *et al* (1986).

The pseudo-inverse learning rule can be written using matrix notation as follows:

$$\mathbf{W} = \mathbf{M}(\mathbf{M}^{T}\mathbf{M})^{-1}\mathbf{M}^{T} \tag{2.8}$$

where **W** is the connection matrix, and **M** is the matrix formed by the training patterns as column vectors.

It is important to keep in mind that most weight matrices of this class will be generated using algorithms that closely approximate the ideal pseudo-inverse weight matrix. Figure 2.5 below, for example, is a distribution of gammas from a weight matrix created by the Blatt & Vergini (1991) learning rule. Also, Abbott states that a point distribution would only be seen in the extreme case of a network of infinite size ($N=\infty$) that has been trained using an infinitely large training set ($P=\infty$).

A typical distribution one might expect to see from weight matrices in this class is shown below in figure 2.5 (below):

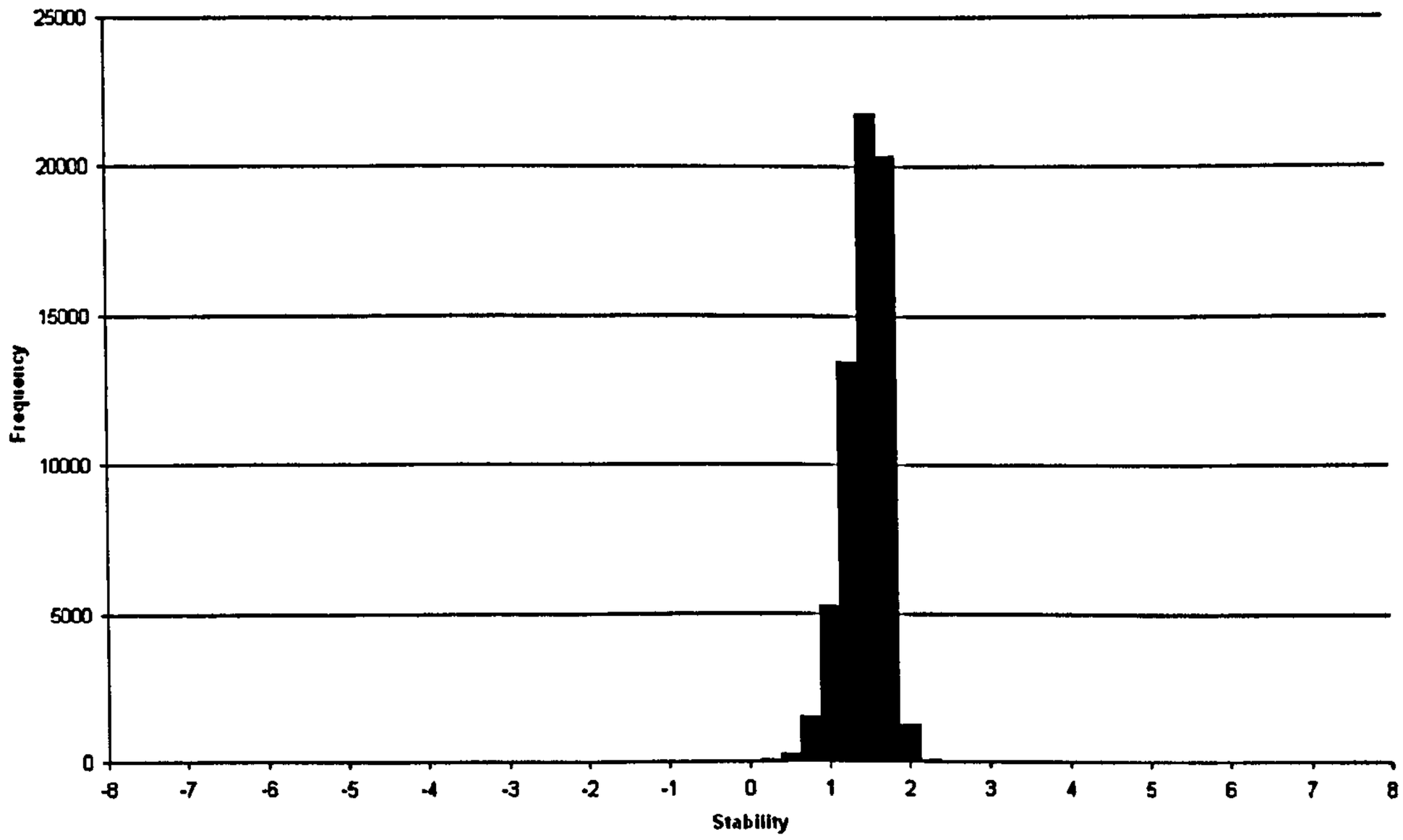


Figure 2.5: Gamma distribution for a network with weights generated using the Blatt & Vergini learning rule.

As expected, it can be seen in figure 2.5 that the distribution, though tight, does not have the zero variance that is stated to be a feature of weight matrices based on the pseudo-inverse. As explained above, this is due to the effect only being seen as $N \rightarrow \infty$.

Weight matrices in this class have a capacity of $\alpha=1$ though at this level of loading linear dependencies within the training patterns is inevitable and the weight matrix becomes the identity matrix. The maximum practical capacity is therefore N-1.

Learning rules that generate weight matrices of this class are: the pseudo-inverse rule, Blatt & Vergini's rule (1991), Iterative Local Learning with Equal Fields (Diederich and Opper, 1987).

Class 3

The third class of weight matrices is known as the *Gardner class* after the extensive work on the space of interactions in neural networks by Gardner (1988).

Distributions of gamma values generated from weight matrices in this class have all gamma values below some critical value K made at least equal to that value.

The fact that the correctness of the aligned local field is not enough to guarantee that the learned patterns will behave as attractors, as mentioned in §2.4.1, is reinforced by Gardner who states that the inequalities:

$$\xi_i^\mu \sum_{j \neq i} W_{ij} \xi_j^\mu > \kappa \tag{2.9}$$

subject to the normalisation condition:

$$\sum_{j \neq i} W_{ij}^2 = 1 \tag{2.10}$$

should imply larger basins of attraction for larger values of K. Equation (2.10) ensures that, by normalising the length of the weight vector, that the aligned local fields are themselves the gamma values for the weight matrix.

So, the larger the value of K of a given weight matrix, the better the attractor performance should be. K is directly related to the capacity, α_{max} (the network loading beyond which not all training patterns will be stable), in that α_{max} will decrease as K gets larger. It is apparent that the reverse must also be true; for a given loading α there exists a maximum value of K, termed K_{max}. This relationship is defined by Gardner as:

$$\alpha = \frac{1}{\int_{-\kappa_{max}}^{\infty} \frac{\exp\left(-\frac{1}{2}t^2\right)}{\sqrt{2\pi}} \left(t + \kappa_{max}\right)^2 dt} \tag{2.11}$$

This relationship is illustrated by the following table showing example values:

Loading (α)	K_{max}
2	0
1	0.5
0.5	1
0.2	2

Table 2.1: The relationship between the loading (α) and the maximum possible lower bound for K for unbiased random patterns.

Table 2.1 shows that the maximum loading for networks of this class should be close to 2 for unbiased random patterns. Gardner concludes that the maximum capacity will increase for correlated patterns. This is an important point and will be discussed in greater detail in chapter 5 where the second phase of this work is introduced.

Distributions of gamma values for weight matrices of this class will look similar to the graph shown in figure 2.6 (below):

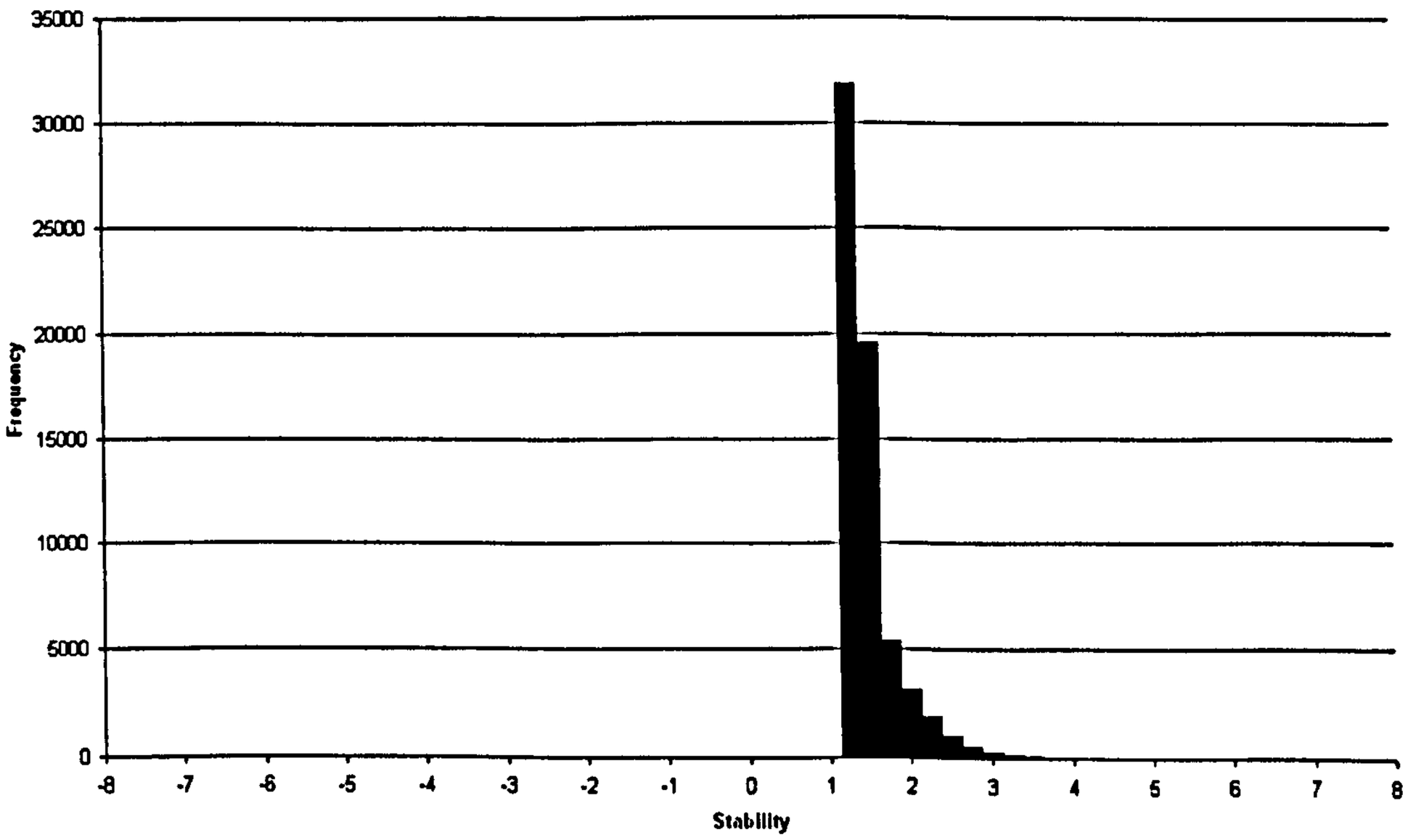


Figure 2.6: Gamma distribution for a network with weights generated using a Gardner class rule.

2.5. Learning Rules

There is a wide range of learning rules applicable to the architecture described in §2.2 and many of these have been mentioned previously in the context of the class of weight matrices that they produce.

What follows is a description of the origin and mechanism of a number of learning rules categorised according to the class of weight matrix that they produce.

2.5.1. Class 1

The following learning rule all produces a weight matrix that is class 1, or Hopfield class.

Storkey learning rule

Storkey's learning rule is an attempt to increase the capacity of the original Hopfield model without some of the sacrifices brought about by some of the more complex algorithms. Storkey's learning rule operates as follows:

$$W_{ij}^{\mu} = W_{ij}^{\mu-1} + \frac{1}{N}\xi_i^{\mu}\xi_j^{\mu} - \frac{1}{N}\xi_i^{\mu}h_{ji}^{\mu} - \frac{1}{N}h_{ij}^{\mu}\xi_j^{\mu} \tag{2.12}$$

where:

$$h_{ij}^{\mu} = \sum_{k=1,k\neq i,j}^{N} W_{ik}^{\mu-1}\xi_k^{\mu} \tag{2.13}$$

where h_{ij}^{μ} is a form of local field or activation at neuron i for pattern μ. The extra terms that Storkey's rule possesses over the Hopfield rule have the effect of partially unlearning the previously presented pattern.

The importance of the Storkey algorithm lies in the fact that it, like the Hebb-inspired learning employed by Hopfield, is immediate in operation. The algorithm is also local with respect to the information it requires to calculate the change to the weights. Some temporal non-locality is present however, as it is apparent from equation (2.13) that the calculation of the local field for the current pattern does not take into account the most recent changes to the weight matrix. In order to implement this rule it is necessary to take the biologically implausible step of pre-calculating the aligned local fields for the entire network for use in training the network on the next pattern at the conclusion of training the current one. The biological implications of this temporary storage of information at a global level exemplify the importance of

temporal locality in a learning rule that is to be a plausible model of neural learning.

2.5.2. Class 2

The following learning rules all produce weight matrices that are class 2, or pseudo-inverse class.

Iterative Local Learning with Equal Fields

Diederich & Opper (1987) propose a learning rule that aims to modify the weights such that the *aligned local fields* of every neuron will eventually become equal to 1 for every training pattern. The aligned local field is defined as $h_i S_i$ at neuron i where h_i is the local field according to equation (2.1). One benefit of forcing the aligned fields of deliberately embedded memories to be equal to 1 is that it provides a method of distinguishing them from spurious states that might be retrieved during recall.

The algorithm begins with a zero weight matrix and proceeds according to:

REPEAT UNTIL ERROR $< \varepsilon$

SET THE NETWORK STATE TO ONE OF THE ξ^{μ}

FOR EACH NEURON i IN TURN

UPDATE THE INCOMING WEIGHTS TO NEURON i ACCORDING TO:

$$\Delta W_{ij} = \frac{\left(1 - h_i^{\mu}\xi_i^{\mu}\right)\xi_i^{\mu}\xi_j^{\mu}}{N} \tag{2.14}$$

where ε is the maximum permitted error across all neurons and patterns and is a small positive constant.

The error is calculated as:

$$E = \sum_{i,\mu}\left|1 - h_i^{\mu}\xi_i^{\mu}\right| \tag{2.15}$$

Equation (2.14) is functionally equivalent to the delta rule (Hertz, Krogh et al., 1991) employed in the training of perceptrons with the learning rate in this case being $1/N$.

The learning rule itself (equation (2.14)) is both temporally and spatially local in its use of information during training. The stopping condition, involving equation (2.15), is clearly non-local. It could be argued that, because of the

dependence of the learning rule on the stopping condition, the algorithm as a whole is non-local.

Blatt & Vergini

Blatt & Vergini (1991) present a learning rule which takes the form of an iterative method for approximating the projection matrix. The training algorithm is guaranteed to find an appropriate weight matrix within a finite number of presentations of each pattern if such a matrix exists.

The algorithm begins with a zero weight matrix and proceeds according to:

FOR EACH PATTERN IN TURN

SET $m = 1$

REPEAT UNTIL ERROR $< \varepsilon$

APPLY THE PATTERN TO THE NETWORK

FOR EACH NEURON IN TURN

UPDATE INCOMING WEIGHTS ACCORDING TO:

$$\Delta W_{ij} = \left(\frac{k^m}{N}\right)\left(\xi_i^\mu - h_i^\mu\right)\left(\xi_j^\mu - h_j^\mu\right) \tag{2.16}$$

SET $m = m + 1$

REMOVE ALL SELF-CONNECTIONS

The error is calculated as:

$$E = \sum_{i,\mu}\left|T - h_i^\mu \xi_i^\mu\right| \tag{2.17}$$

where T is the desired threshold value for the aligned local field.

As this learning rule generates a weight matrix approximating that generated by the pseudo-inverse rule but possessing a non-zero leading diagonal the final step of removing the self connections (W_{ii}'s) must be taken in order to guarantee convergence upon the stored patterns.

Blatt & Vergini present a formula for calculating the minimum number of presentations of the training set to perform in order to achieve aligned local fields with values of at least T. The number of presentations, V, is calculated as being the smallest integer conforming to:

$$V \geq \log_k\left(\frac{N}{(1-T)^2}\right) \tag{2.18}$$

where k and T are real valued constants such that $1 < k \leq 4$ and $0 \leq T < 1$. k is referred to as the *memory coefficient* of the network; the larger it is, the fewer steps are required to train the network.

For efficiency, this work employs a version of the learning rule which pre-calculates the number of training set presentations required by using equation (2.18). This removes the need for a computationally intensive test for the aligned local field being correct to be performed at each iteration.

The Blatt & Vergini learning rule has the advantage of being local in its use of information but crucially it is also incremental. Remarkably, the addition of further patterns to the network can be made without harm or disturbance to the stored patterns already present.

2.5.3. Class 3

The following learning rules all produce weight matrices that are class 3, or Gardner class.

Gardner's Rule

Gardner (1988) proposed a learning rule which, provided such a solution exists, will find a solution to equation (2.9) which forces all gamma values above some specified value, K.

The algorithm begins with a zero weight matrix and proceeds according to:

REPEAT UNTIL ALL GAMMA VALUES ARE CORRECT

FOR EACH PATTERN μ IN TURN

FOR EACH NEURON i IN TURN

IF $\gamma_i \leq K$ THEN

UPDATE THE INCOMING WEIGHTS ACCORDING TO:

$$\Delta W_{ij} = \frac{1}{N} f\left(h_i^\mu\right) \xi_i^\mu \xi_j^\mu \tag{2.19}$$

where h_i is the local field defined by equation (2.1) and $|W|_i$ is the length of the incoming weight vector at neuron i.

Gardner offers two choices for the function $f\left(h_i^\mu\right)$:

i) The perceptron algorithm:

$$f\left(h_i^\mu\right) = 1 \tag{2.20}$$

which is guaranteed to converge upon a solution, if one exists, in a finite number of training steps.

ii) The relaxation algorithm (Abbot and Kepler, 1989):

$$f\left(h_i^\mu\right) = \lambda\left(\kappa - h_i^\mu\right)|W|_i \tag{2.21}$$

which, if a solution exists, will converge for $0 < \lambda \leq 2$ and according to Gardner, is most efficient for λ=2.

This algorithm is not considered for implementation in this work due to the need to calculate the length of the incoming weight vector at each weight change. This is considered, in this work, to be outside the spirit of local computation.

Iterative Local Learning

Diederich & Opper (1987) and Forrest (1988) both propose a learning rule that is similar in operation to the perceptron rule (Rosenblatt, 1958). This iterative scheme aims to drive the local fields for each training pattern to the correct side of $+T$ or $-T$ is appropriate. This goal can be formalised in terms of the aligned local fields as follows:

$$h_i^\mu \xi_i^\mu \geq T \qquad \text{for all } i, \mu \tag{2.22}$$

The algorithm begins with a zero weight matrix and proceeds according to:

REPEAT UNTIL LOCAL FIELDS ARE CORRECT

SET THE NETWORK STATE TO ONE OF THE ξ^μ

FOR EACH NEURON i IN TURN

IF $h_i^\mu \xi_i^\mu < T$ THEN UPDATE THE INCOMING WEIGHTS TO NEURON i ACCORDING TO:

$$\Delta W_{ij} = \frac{\xi_i^\mu \xi_j^\mu}{N}$$

If one exists, this learning rule will converge upon a suitable weight matrix for which all the trained patterns are guaranteed to be stable. This rule is very similar to the Gardner rule (previous page) with:

$$f\left(h_i^\mu\right) = 1 \tag{2.23}$$

Iterative Local Learning is fully local in its use of information and, as its name implies, is iterative in operation.

A symmetric version of this rule exists (Gardner, 1988) in which an weight change at each W_{ij} is replicated at W_{ji}. This has the advantage of ensuring simple updates dynamics as described earlier in §2.3.

Krauth & Mezard's Learning Rule

Krauth & Mezard (1987) propose a modification to the iterative local learning rule (Diederich and Opper, 1987). This rules differs from the original in that at each pattern presentation opportunity the pattern with the smallest aligned local field is selected to be the presented. This is in contrast to the undefined presentation order of iterative local learning.

This learning rule will produce a value of K (Gardner, 1988) that tends towards K_{max} as the training threshold, T, increases.

The algorithm begins with a zero weight matrix and proceeds according to:

REPEAT UNTIL LOCAL FIELDS ARE CORRECT

FOR EACH NEURON i IN TURN

SELECT THE PATTERN WITH THE LOWEST ALIGNED LOCAL FIELD AT THIS UNIT

THEN UPDATE THE INCOMING WEIGHTS TO NEURON i ACCORDING TO:

$$\Delta W_{ij} = \frac{\xi_i^\mu \xi_j^\mu}{N} \tag{2.24}$$

The advantage that this learning rule has over iterative local learning is that it is capable of finding the optimal value for K when using a sufficiently high threshold.

2.5.4. Other Learning Rules

Other learning rules exist beyond those described in the sections above. Davey *et al.* (2002) identify two worthy of note. The first of these is an alternative technique for finding a weight matrix of Gardner class proposed by Athithan (1995). In this work Athithan approaches the training of the network as an optimisation problem solvable by linear programming techniques. The further investigation of this mathematical approach is outside the scope of this project.

The second rule identified is proposed by Plakhov and Semenov (1994). Their technique initialises the weight matrix according to one-shot Hebbian learning and proceeds to train further by applying random patterns to the network and 'unlearning' them. Again, further investigation of this rule is outside the scope of this work.

2.6. Further Variations

The set of learning rules described in §2.5 is not intended to be exhaustive but rather to encompass a range of what might be considered the most interesting algorithms with which to experiment. Work has been conducted on improving the performance of networks by changing them further upon conclusion of the weight modification process and one of the techniques employed in doing this is described below.

2.6.1. Modification of Neuron Thresholds

It was seen in §2.3.1 that the original Hopfield network was made up of neurons with thresholds set to zero. Modifying the thresholds of a network's neurons would be an obvious scheme through which the network performance might be improved upon.

Schultz (1995) proposes a system whereby the threshold of a neuron is set to a value exactly halfway between the values of the largest negative and smallest positive local fields taken across all patterns. The motivation behind this technique is to maximise the 'slack' over the set of training patterns. The term 'slack' is best described using Schultz's own example.

Consider, for example, a neuron in a trained network. The local field for each of four training patterns is: -3, -1, 5, and 7. The desired output values at that neuron for the training patterns are: -1, -1, 1, and 1 respectively. It can be seen from the local field of -1 that if the neuron threshold is set at zero then only a small amount of corruption (1 unit) in the pattern presented for recall can cause that neuron to output the incorrect value (+1) as the local field is pushed above the threshold.

Schultz suggests a value of 2 would be more appropriate for the threshold so as to maximise the separation between the positive and negative local fields with the smallest magnitudes. This provides greater error correction capability for patterns that might otherwise be particularly susceptible to failing at low levels of corruption. It is this separation between the local fields that is termed the 'slack'.

The new threshold value is given by:

$$\theta_i = \frac{h_i^+ + h_i^-}{2}, \quad \text{where} \quad \begin{aligned} h_i^+ &= \min\left\{h_i^p \mid h_i^p \geq 0\right\} \\ h_i^- &= \max\left\{h_i^p \mid h_i^p < 0\right\} \end{aligned} \tag{2.25}$$

Results of performance analyses of networks employing this technique may be found in Davey *et al* (2002).

Buckingham and Willshaw (1993) examine in detail a number of threshold-setting strategies both simple and complex based, in part, on Marr's (1971) proposal that the value of the threshold should depend on a neuron's input activity. The full range of techniques is too wide to cover here and the deployment of them outside the scope of this project but both Buckingham and Willshaw and Schultz (1995), mentioned above, present opportunities for investigating further improving the performance of the networks studied within this work.

3. NETWORK PERFORMANCE ANALYSIS TOOLS

3.1. Introduction

To be able to compare the relative performance of the networks and learning rules used throughout this work it was necessary to have a robust set of analysis tools. This chapter presents a description of the performance measures used throughout this investigation.

The measures are: pattern stability, capacity, training time, and attractor performance.

3.2. Performance Metrics

3.2.1. Pattern Stability

The simplest test is that of pattern stability. A network is placed in a start state known to correspond exactly to one of the patterns the network has been trained upon. If the network state, upon update of all neurons in accordance with the network dynamics, has moved from that initial state to some other, then the pattern forming the start state is deemed not to be stable.

The presence or absence of unstable patterns at a particular loading assists in determining the capacity of a network (described below) as well as providing an indication of the speed at which a network's memory of a set of training patterns fails.

3.2.2. Pattern Load

While not in itself a performance metric, the loading placed on the network is important in the calculation of metrics such as capacity, described below. The loading on a network, denoted by the symbol , is calculated according to:

$$\alpha = \frac{P}{N} \tag{3.1}$$

where P is the number of patterns in the training set and N is the size of the network.

3.2.3. Capacity

The capacity of a network represents the maximum loading that can be placed upon the network with all the patterns remaining stable. The capacity, denoted by $_{max}$, is calculated in the same manner as the loading:

$$\alpha_{max} = \frac{P}{N} \tag{3.2}$$

where P is the number of patterns in the training set and N is the size of the network.

The capacity can be determined using the pattern stability measure described above. If the number of patterns to be learnt by a network is gradually increased and the network retrained each time, the last loading at which all the patterns are stable provides a maximum value for P in equation (3.2).

3.2.4. Training time

The training time of a network is reported in terms of the number of iterations through the training set that is required for the network to be fully trained. This measure is only applicable to networks with those learning rules where multiple presentations of patterns are required.

3.2.5. Attractor Basin Size

The attractor basin size is a measure designed to indicate the recall ability of a network when given, as its start state, a corrupted version of one of the trained patterns. The stored pattern is considered to be acting as a final state to which the evolving state of the network is attracted through the action of the update dynamics.

The attractor performance of an associative memory can be defined in terms of the radii of the basins of attraction of the stored patterns. The analogy is often made between a marble started high on the wall of a basin rolling to a standstill at the basin's base and the relaxation of networks with Hopfield-type dynamics into a state of low energy. The radius of the basin of attraction is correspondingly analogous to how far, in Hamming distance, one can move the start state of the network away from a stored pattern and still have the pattern recalled correctly by the network.

As the state space of a network forms a discrete N-dimensional hypercube with the states of the network at the vertices it is somewhat incorrect to think of attractor performance in this 3-dimensional, continuous fashion. It does however, serve as a useful visualisation of the activity of the network (Hertz, Krogh et al., 1991).

3.2.5.1. The Kanter and Sompolinksy Attractor Basin Measure

Kanter and Sompolinsky (1987) devised a technique for measuring the size of the basins of attraction. This measure is effectively an average of the basin sizes for all the patterns embedded in the network. Based on the method of gradually increasing the corruption of a known stored pattern and attempting recall using the corrupted pattern as a network's initial state (Hopfield, 1982), the calculation is performed as follows:

For a given set of P patterns, a network start state is chosen from the training data. The first mN bits of the pattern selected are fixed to be equal to those of the original pattern. The value, m, represents the proportion of the start state that is to remain the same as the stored pattern and is termed the *overlap*. For example, consider a simple network of 10 units:

We assume some start state:

```
-1   1 -1   1 -1   1 -1   1 -1   1
```

If we begin with a high overlap, or a value of $m = 0.9$, then our randomly corrupted state might be:

```
-1   1 -1   1 -1   1 -1   1 -1 -1
```

The fixed portion of the states is shown using bold text. This new pattern is applied to the network and the network permitted to update until it converges upon a pattern. If the updated pattern is equal to the source pattern i.e. the uncorrupted original, then the successful start state is recorded. This process is repeated for a number of different initial states derived each time from a known stored pattern. If all the start states tested converge correctly then the current value m is recorded and denoted m_0. The value of m is then lowered and the process begins again. m_0 always represents the furthest successful point tested so far at which all, or most, of the sample states flow to the original patterns.

The above procedure is repeated until an average m_0, calculated over different sets of patterns, has been obtained. The number of pattern sets is arbitrary though the larger the sample set the greater the accuracy of the final measure. A value the size of the basin of attraction, R, can now be defined as:

$$R = 1 - \langle m_0 \rangle \tag{3.3}$$

Kanter and Sompolinsky note that for small values of , where R is close to 1, the effect that the size of the network has on the result is not insignificant. In a theoretical network of infinite N, the distance between patterns is very large - there is no interference between them. At low loadings in a finite size network one would expect the R value to be close to the maximum of 1. At these low loadings however, the effect that interference might have on the result must be taken into account. As loading increases, the importance of the effect of pattern interference falls by comparison with other factors affecting attractor performance such as spurious memories.

To partially compensate for the interference, Kanter and Sompolinksy refine equation (3.3) so that it becomes:

$$R = \langle\langle \left[\frac{1 - m_0}{1 - m_1} \right] \rangle\rangle \tag{3.4}$$

where m_1 is the largest overlap of the initial states with the rest of the patterns and is calculated using the set of corrupted patterns stored from the first procedure. Implementing this involves a record being made of the largest overlap each corrupted pattern has with the patterns in the training set that are not the original source pattern and these values become the m_1's for use in equation (3.4)

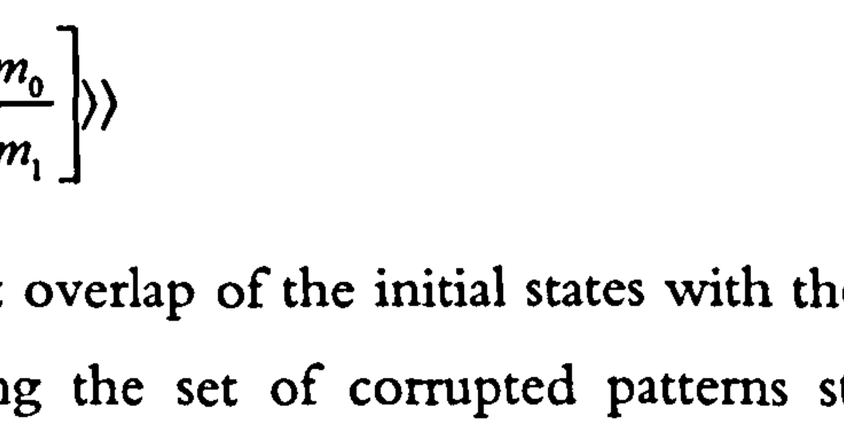


Figure 3.1: A stylised representation of the attractor basin for an imaginary pattern p1. The next nearest pattern to the last successful convergence point is pattern p2. The dashed line represents just one successful sample.

Figure 3.1 (previous page) shows the various values measured during the calculation of the Kanter and Sompolinksy measure. An ideal attractor basin around pattern p1 is shown. The next nearest pattern to the last successful sample state is denoted p2.

Note that the refined version of the measure no longer acts as a direct measure of the attractor basin size but rather becomes a relative measure that takes into account the proximity of the patterns to each other. This has the advantage of producing a single value for the 'goodness' of a network's attractor performance based on some knowledge of the dispersal of the patterns in the pattern space. The disadvantage is that the absolute measure of the basin size in terms of the proportion of corrupt bits correctable has been lost due to the normalisation in equation (3.4).

3.2.5.2. Modified Kanter and Sompolinsky Measure

Two aspects of the original Kanter and Sompolinksy measure were modified to produce the version employed in this work.

Firstly, when choosing a number of bits to fix in order to produce an overlap with a stored pattern, the original measure always fixed the first mN bits. Fixing the same bits each time a sample pattern is generated causes the sample patterns to be rooted in the same area of the state space. To counteract this, the bits which are to be fixed in each sample pattern are chosen at random.

Secondly, assigning the unfixed bits randomly to be equal to +/-1 does not guarantee a sample pattern to be exactly the required distance away from the stored pattern. The modification implemented to resolve this was to invert the unfixed bits, thus resulting in a sample pattern that is exactly the specified distance away each time.

Inverting rather than flipping the unfixed bits also addresses a disparity in the way the pattern overlaps are measured for the values m_0 and m_1. m_0 is regarded as being the overlap of the last successful sample state with the source pattern as a proportion of the total length of the pattern. This fails to take into account that when randomly flipping the unfixed portion of a sample pattern, half the flipped bits will, on average, have the same value as in the original pattern. Therefore, the average overlap, when taken over all the samples, will in fact be $m_0 + 0.5\ (1 - m_0)$. When calculating m_1, the value used is genuinely the

overlap with the next nearest pattern. Inverting rather than flipping the unfixed bits ensures the fact that both overlaps are accurate.

3.2.5.3. Comprehensive Basin Analysis Measure

The comprehensive basin analysis measure (CBAM) was developed as part of this work in order to address an issue with the Kanter and Sompolinsky measure described above. The Kanter and Sompolinksy measure produces a single comparative value indicating the overall recall quality of the network but loses information related to the absolute attractor performance of the individual stored patterns.

The new measure is based on the notion of having a *reference basin* for each stored memory. The reference basin of a pattern is calculated as being one half the Hamming distance between it and the pattern nearest to it. As the inverse of one of the intentionally stored patterns may be closer than one of the stored patterns themselves then they too are compared against. The reference basin is used as a reference because half the distance between a pattern and its nearest neighbour is equal to the largest basin size we might reasonably expect for that pattern. This is the case because the reference basin represents the furthest one can move away from a pattern before becoming closer to some other.

Once the reference basins have been calculated for the set of stored patterns, a predetermined number of *walks* are taken from each of the stored patterns. This is done in a similar way as with the Kanter and Sompolinsky measure described above. Sample patterns are generated at increasing Hamming distance from the stored pattern and the network allowed to update in order to determine whether or not it can recall the original source pattern. A number of these walks are undertaken and the maximum distance attained is recorded.

Having acquired a set of samples for each of the stored patterns, each maximum distance achieved is normalised with respect to the reference basin size for the corresponding stored pattern. It is not unlikely that non-random patterns will be unevenly distributed in the state space. This normalisation provides an indication of the attractor performance while taking into account the proximity of the stored patterns to each other. In this respect the new measure operates in a similar manner to Kanter and Sompolinsky's.

The mean of each pattern's set of normalised values provides a value representing how well that pattern is performing as an attractor compared with the best we might expect of it given the other patterns in the vicinity. The variance of the set of normalised values provides an indication of how skewed the basin of attraction is. For instance, if the other patterns are evenly distributed in the state space then we might expect that the distance achieved during each walk would be similar and thus the variance of the samples would be small. If however, the patterns are correlated to any degree then each walk may well result in quite a different degree of success. This would result in a higher variance of the samples and would indicate a more uneven attractor basin.

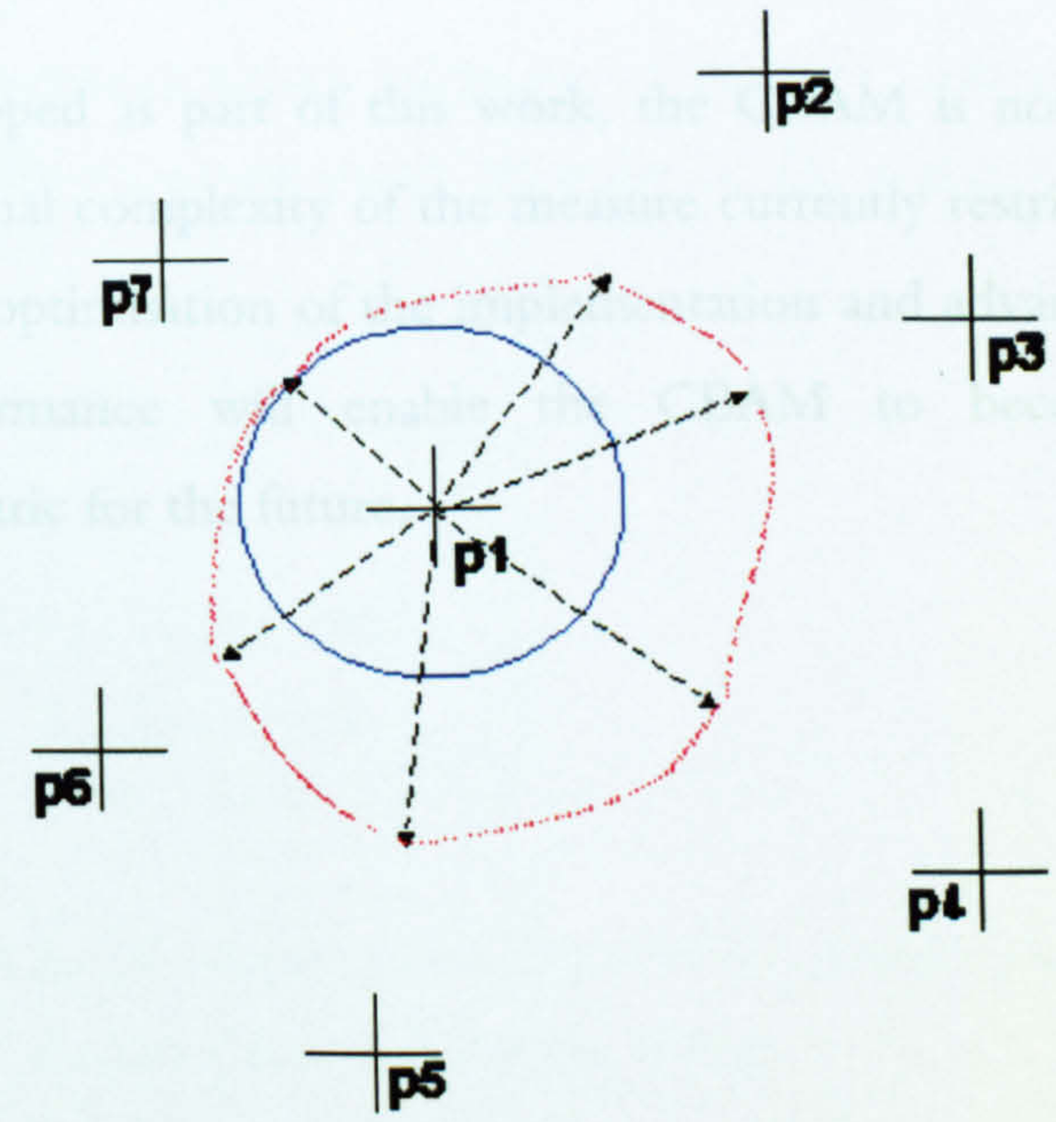


Figure 3.2: A stylised representation of the reference basin (blue line) for an imaginary pattern p1. Sample walks are shown as dashed lines. The resulting attractor basin (red line) can be seen to be non-circular.

Figure 3.2 portrays the relationship between the reference basin, denoted by a blue line, and the resulting attractor basin which is shown using a red line. The reference basin is of a diameter equal to one half the distance between pattern p1 and the next nearest pattern, p7. The sample walks represented by the dashed lines can be seen to be of different lengths. It is the variance of these lengths that indicates the 'skewedness' of the attractor basin. It should be kept in mind that figure 3.2 is a 2-dimensional representation of what would an *N*-dimensional space and serves as a visualisation aid only.

The double average of the mean normalised values over all the stored patterns provides a value analogous to the final Kanter and Sompolinksy measure defined by equation (3.4).

As well as simply providing more information about the attractor behaviour of a network than the Kanter and Sompolinsky measure, the CBAM also results in a measure that is directly translatable into an absolute figure representing the mean number of bits of pattern corruption that a network is capable of correcting.

The finite size effects that are taken into account in Kanter and Sompolinsky's measure are implicitly addressed through the use of the reference basin concept.

Although developed as part of this work, the CBAM is not used within it. The computational complexity of the measure currently restricts its usefulness. It is hoped that optimisation of the implementation and advances in computer hardware performance will enable the CBAM to become a valuable performance metric for the future.

4. PERFORMANCE OF FULLY CONNECTED NETWORKS

4.1. Introduction

It is widely known that the original Hopfield network suffers from low capacity and unsuitability for correlated patterns. As was seen in chapter 2 however, it is by no means the only learning rule which is available for training networks of the Hopfield-type.

The purpose of this chapter therefore, is to present the results of analysing the performance of a number of higher performance learning rules. This analysis was important to undertake as the identification of a suitable learning rule was critical for later work.

Learning rules that generate weight matrices that belong to one of the three Abbot classes described earlier are examined; all three of the classes are represented to various degrees.

The learning rules used are:

Class 1

Hopfield (Hopfield, 1982)

Storkey (Storkey, 1997)

Class 2

Iterative Local Learning with Equal Fields (Diederich and Opper, 1987)

Blatt & Vergini (Blatt and Vergini, 1991)

Class 3

Iterative Local Learning (Diederich and Opper, 1987)

Symmetric Local Learning (Gardner, 1988)

Krauth & Mezard (Krauth and Mezard, 1987)

Networks employing these learning rules are assessed with respect to the time taken to store a set of training patterns, the stability of the learnt patterns, and the attractor performance of the network.

The training data comprises two classes of randomly generated bipolar patterns. The first of these is unbiased, the second constructed with a bias towards +1 of 0.8. The networks used are 100 neurons in size (N=100). The values plotted are the mean of five experimental runs.

Where parameters controlling the training process may be used they were set as follows:

Iterative Local Learning with Equal Fields: the minimum error on the aligned local field was required to be <= 0.1. Preliminary results determined this value to be the most appropriate choice from a performance versus training time point of view.

Blatt and Vergini: the memory coefficient, k, was equal to 4, the maximum permitted value. The training threshold, T, was equal to 0.99 where T must be < 1 for the Blatt and Vergini rule.

Iterative Local Learning, Symmetric Local Learning, and Krauth and Mezard: the value of the training threshold, T, was again determined by preliminary results to be best made equal to 10.

4.2. Training Time

The networks' training times were measured in terms of the number of presentations of the training set that were required before the patterns where learnt. In the case of the Krauth and Mezard learning rule, where the patterns are not presented an equal number of times, the time reported is a pseudo-iteration calculated as:

$$\text{Pseudo-iterations} = \frac{\text{Total number of presentations made}}{\text{Number of patterns}}$$

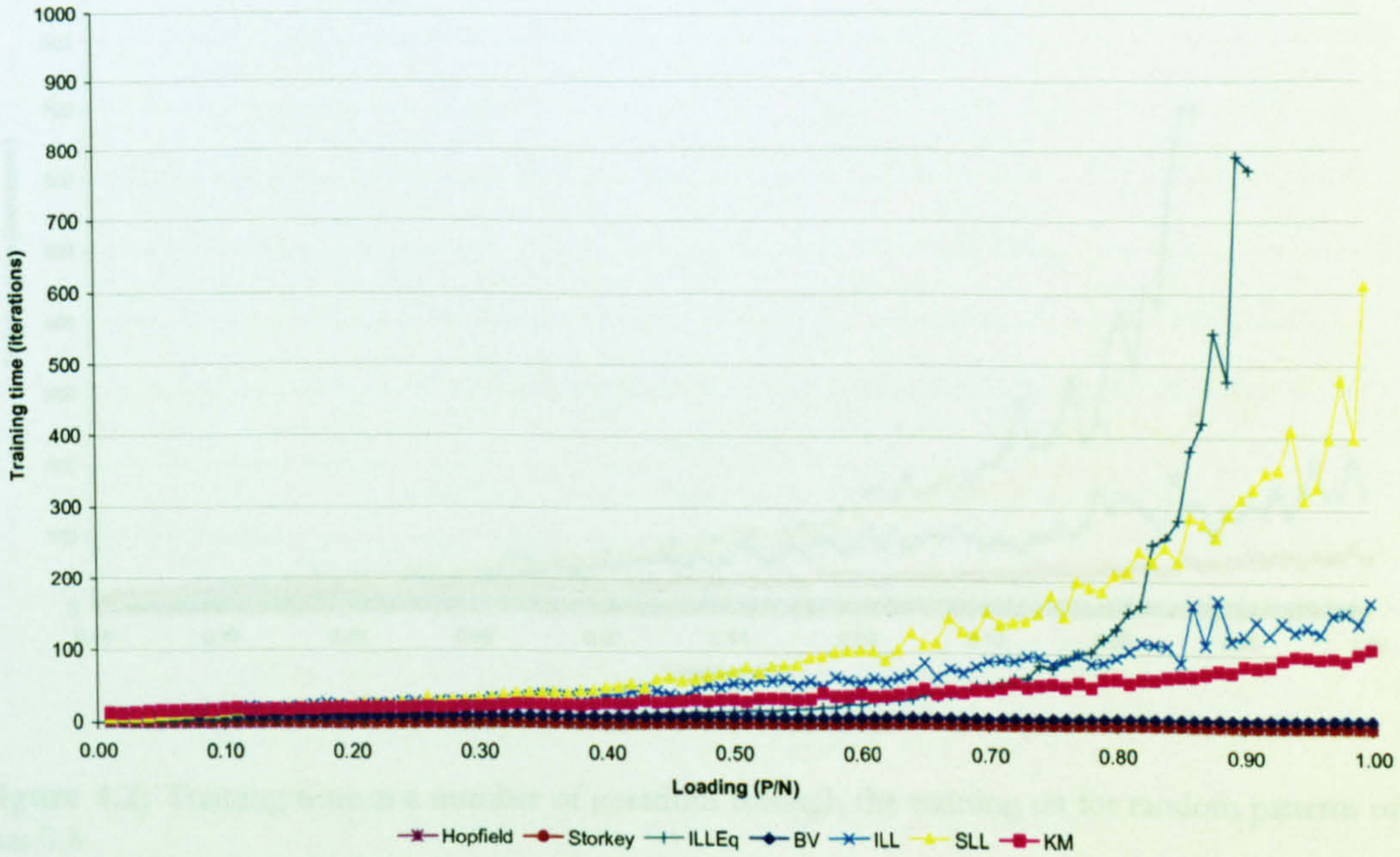


Figure 4.1: Training time as a number of iterations through the training set for random patterns of bias 0.5.

Figure 4.1 (previous page) shows the number of iterations required to learn increasing loadings of random unbiased training patterns. It is very clear that the fastest learning rules are the one-shot, class 1 rules, Hopfield and Storkey. These two rules require only a single iteration through the training set. The next quickest learning rule is that of Blatt and Vergini (BV), requiring a constant 10 iterations through the training set regardless of the pattern load.

The Krauth and Mezard (KM) learning rule would appear to be next fastest, certainly at higher loadings. The KM rule is however much more computationally expensive than any of the other learning rules, requiring a check before each presentation for the pattern containing the bit with lowest aligned local field. In real time, the KM rule is slower than even Iterative Local Learning with Equal Fields at producing a weight matrix.

Iterative Local Learning with Equal Fields (ILLEq) is, therefore, the next fastest learning rule but only up to a loading of α=0.75. Iterative Local Learning (ILL) becomes quicker at higher loadings.

Finally, the symmetric version of Iterative Local Learning (ILL) proves to be the slowest learning rule (barring KM) at low loadings but falling somewhere between ILL and ILLEq as the loading gets to α=0.83.

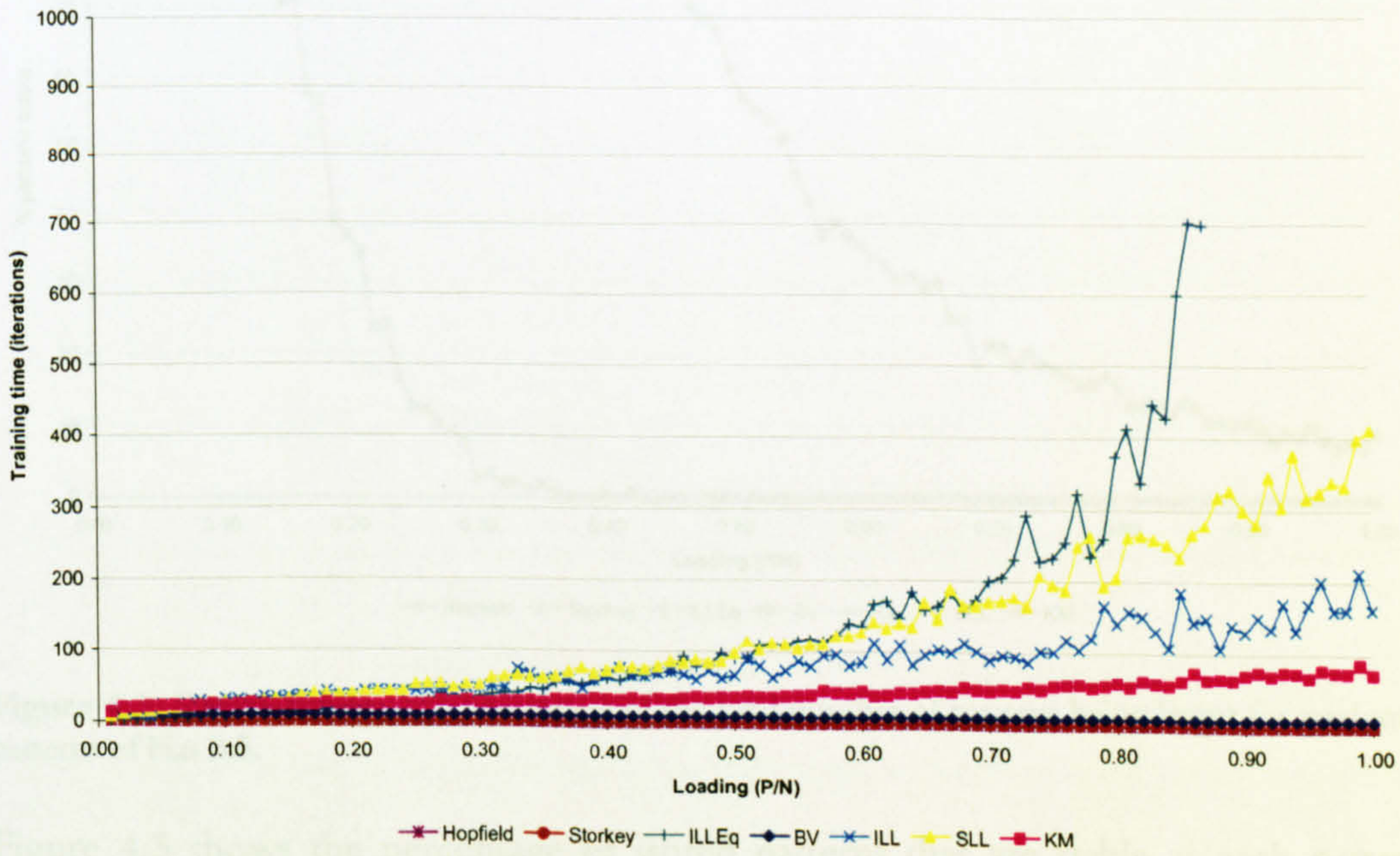


Figure 4.2: Training time as a number of iterations through the training set for random patterns of bias 0.8.

Figure 4.2 shows the number of iterations required to learn increasing loadings of random training patterns with bias 0.8. The order in which these networks are placed with respect to their relative training times is largely the same as when the networks were trained using unbiased random data.

The training times are consistently longer for networks learning biased data in all but only particular instance. The KM learning rule appears to take fewer pseudo-iterations to learn a biased training set than it does an unbiased one. This only occurs at high loading however. The SLL rule appears to be a more appropriate choice for biased patterns than it was for unbiased patterns. Comparing its training time with that of ILLEq shows it to become competitive at a much lower loading than was seen for the unbiased data.

4.3. Pattern Stability

The pattern stability is measured as the percentage of the training patterns that, when applied to the network as a start state, result in the network remaining in that state upon application of the update dynamics.

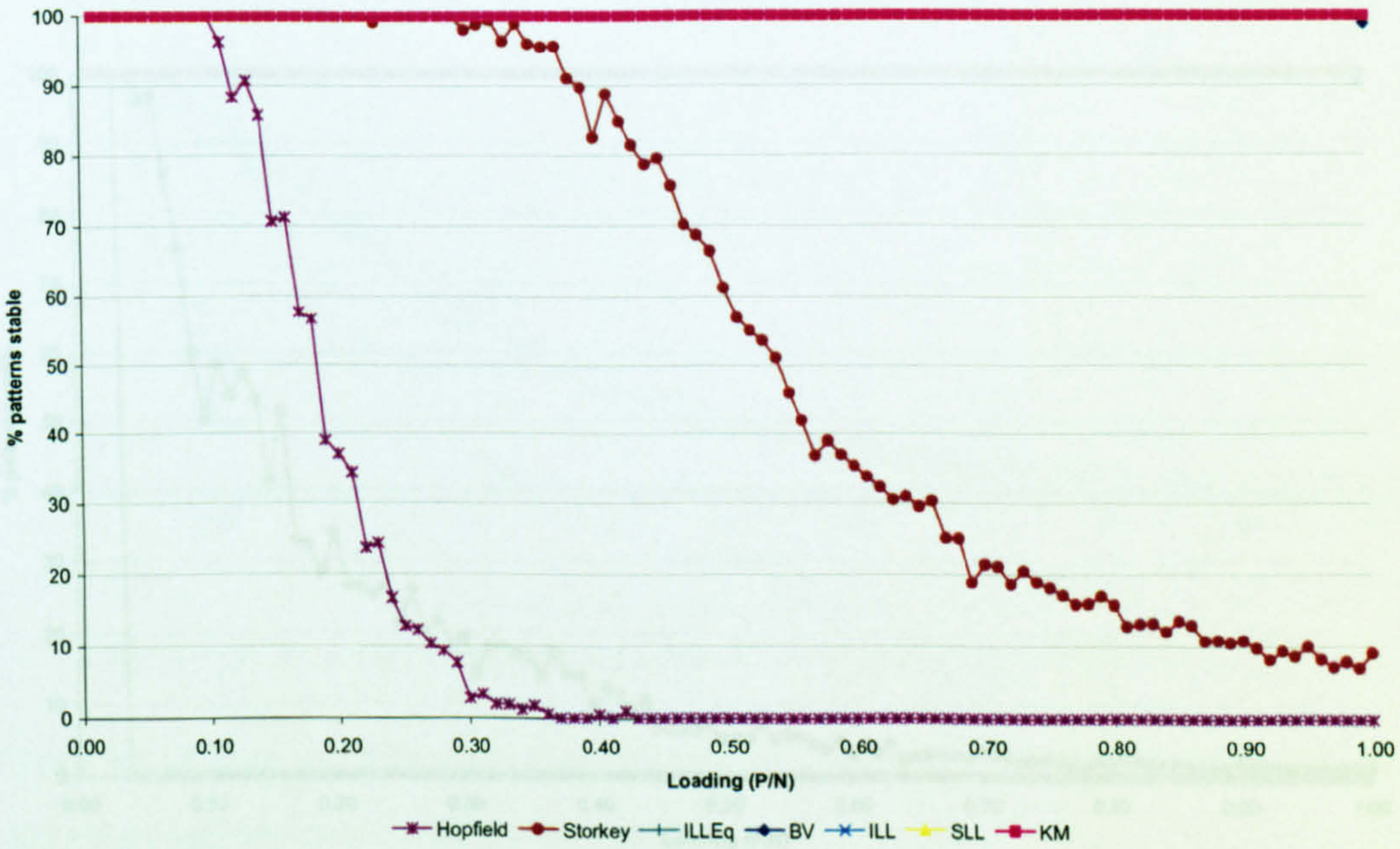


Figure 4.3: Pattern stability as a percentage of the total number of patterns being learnt for random patterns of bias 0.5.

Figure 4.3 shows the percentage of stored patterns that are stable at each pattern load for unbiased random training patterns. The striking feature of this graph is the number of learning rules capable of storing the maximum 100 patterns. It is the two class 1 learning rules, Hopfield and Storkey, that result in less than near 100%

stability at all loadings. The only other learning rule to exhibit any kind of stability failure is the Blatt and Vergini rule. The BV rule loses stability slightly at the maximum loading of α=1 (100 patterns). This is not surprising as the BV rule is a pseudo-inverse rule approximator which, as detailed in §2.4.1, has a maximum capacity of α_{max}=1 (N patterns). It is interesting to note that no instability is apparent for the other class 2 learning rule, Iterative Local Learning with Equal Fields. All the class 3 rules, the Gardner-type algorithms exhibit 100% stability at all loadings. Again, this is not surprising as their notional maximum capacity is α_{max}=2 ($2N$ patterns).

It is worth making a point about the two learning rules that do result in a significant drop in pattern stability with loading. The Storkey rule is a direct modification of the Hopfield rule and was designed to improved performance without compromising the speed of learning of the original. The Storkey rule appears capable of storing twice as many patterns as the Hopfield rule for the same level of pattern stability. As the Storkey rule retains the one-shot nature of the Hopfield rule it would seem to be an excellent choice for learning low numbers of patterns.

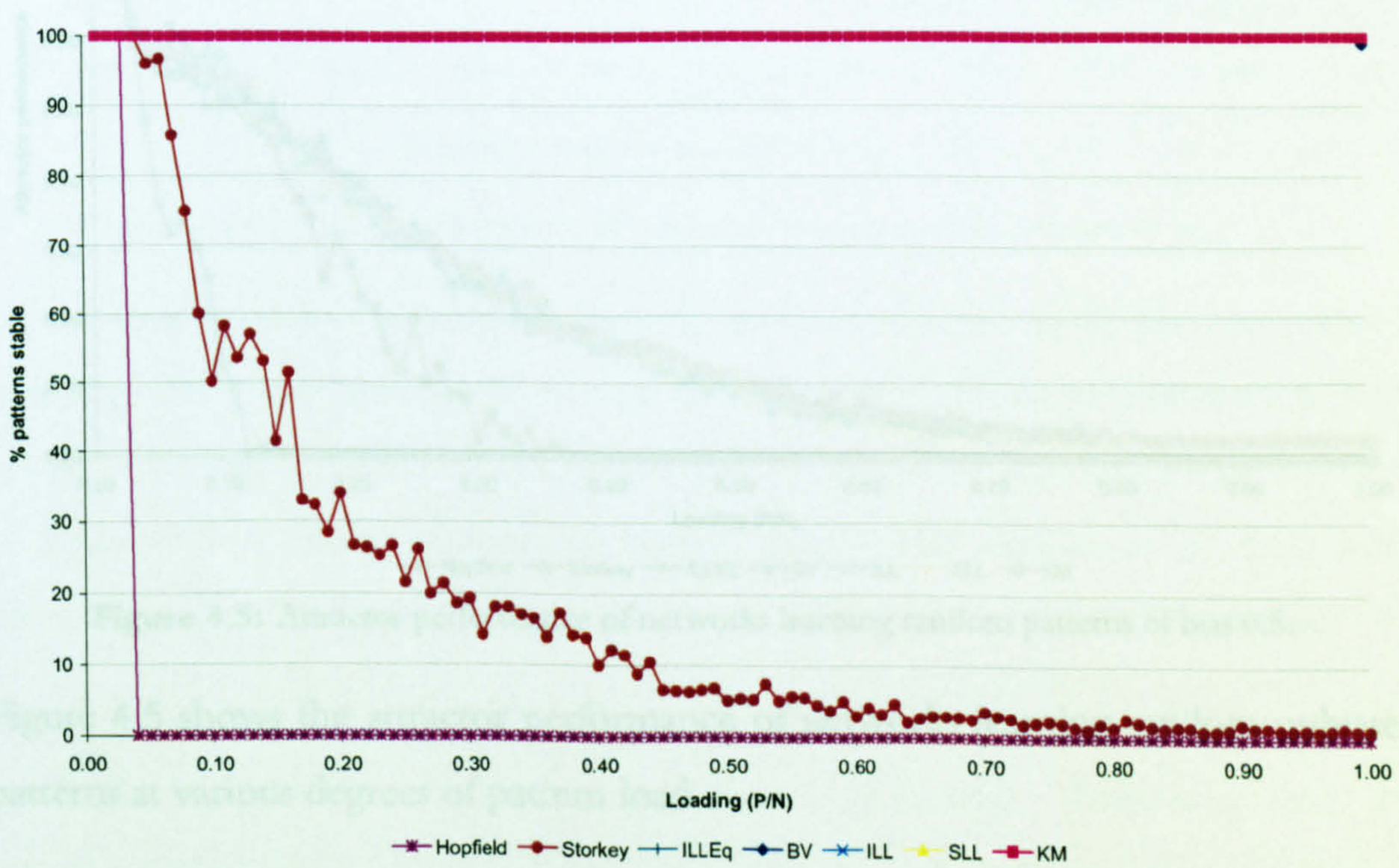


Figure 4.4: Pattern stability as a percentage of the total number of pattern being learnt for random patterns of bias 0.8.

Figure 4.4 shows the percentage of stored patterns that are stable at each pattern load for random training patterns of bias 0.8. The high level of stability exhibited

by the class 2 and 3 learning rules for unbiased patterns is again present for biased data.

The weakness of the class 1 learning rules against correlated patterns is clearly evident. The Storkey rule does however manage to perform significantly better than the Hopfield; the difference between the two rules is much greater in the case of biased data. Networks trained using the Hopfield rule lose all pattern stability at around a loading of α=0.04 (4 patterns). The Storkey rule manages to retain some pattern stability at up to the maximum loading of α=1 (100 patterns).

4.4. Attractor Performance

The attractor performance of the network at various pattern loads was measured using the modified version of the Kanter and Sompolinksy (1987) measure described in §3.2.5.1.

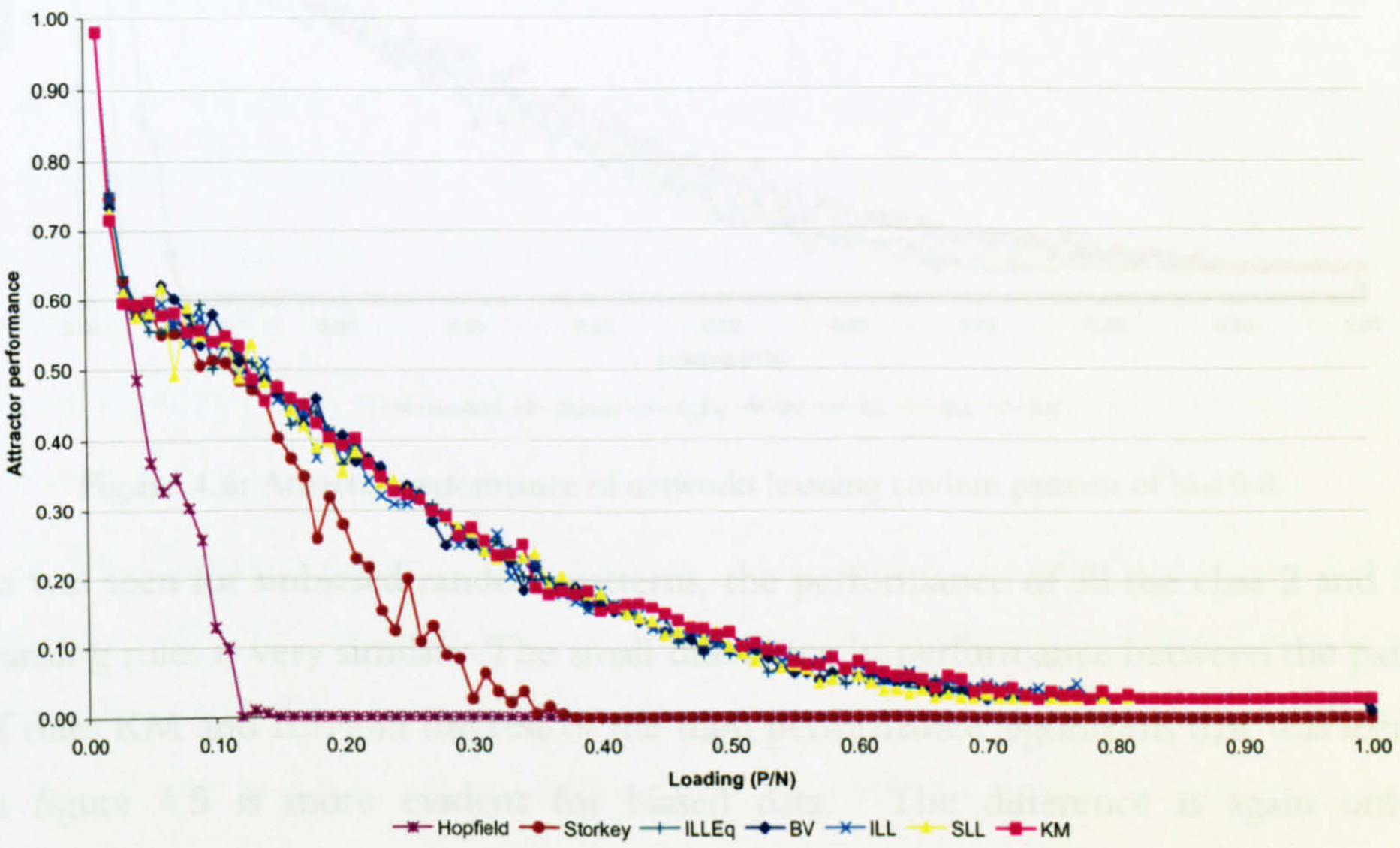


Figure 4.5: Attractor performance of networks learning random patterns of bias 0.5.

Figure 4.5 shows the attractor performance of networks learning random unbiased patterns at various degrees of pattern load.

It is again clear that the two class 1 learning rules, Hopfield and Storkey, are inferior to any of the class 2 or 3 rules and, as was seen to be the case for pattern stability, the Storkey rule outperforms the Hopfield at the same pattern load.

The class 2 and 3 learning rules all result in very similar performance at each pattern load level. Closer examination reveals a possible slight edge for the Krauth and

Mezard and Iterative Local Learning rules. This only appears to be the case between loadings of α=0.40 (40 patterns) and α=0.80 (80 patterns).

Figure 4.6 shows the attractor performance of networks learning random patterns of bias 0.8 at various degrees of pattern load.

Once again the Hopfield and Storkey learning rules perform poorly when asked to store correlated patterns. The Storkey rule outperforms Hopfield but as the performance of both rules is poor this makes little real difference.

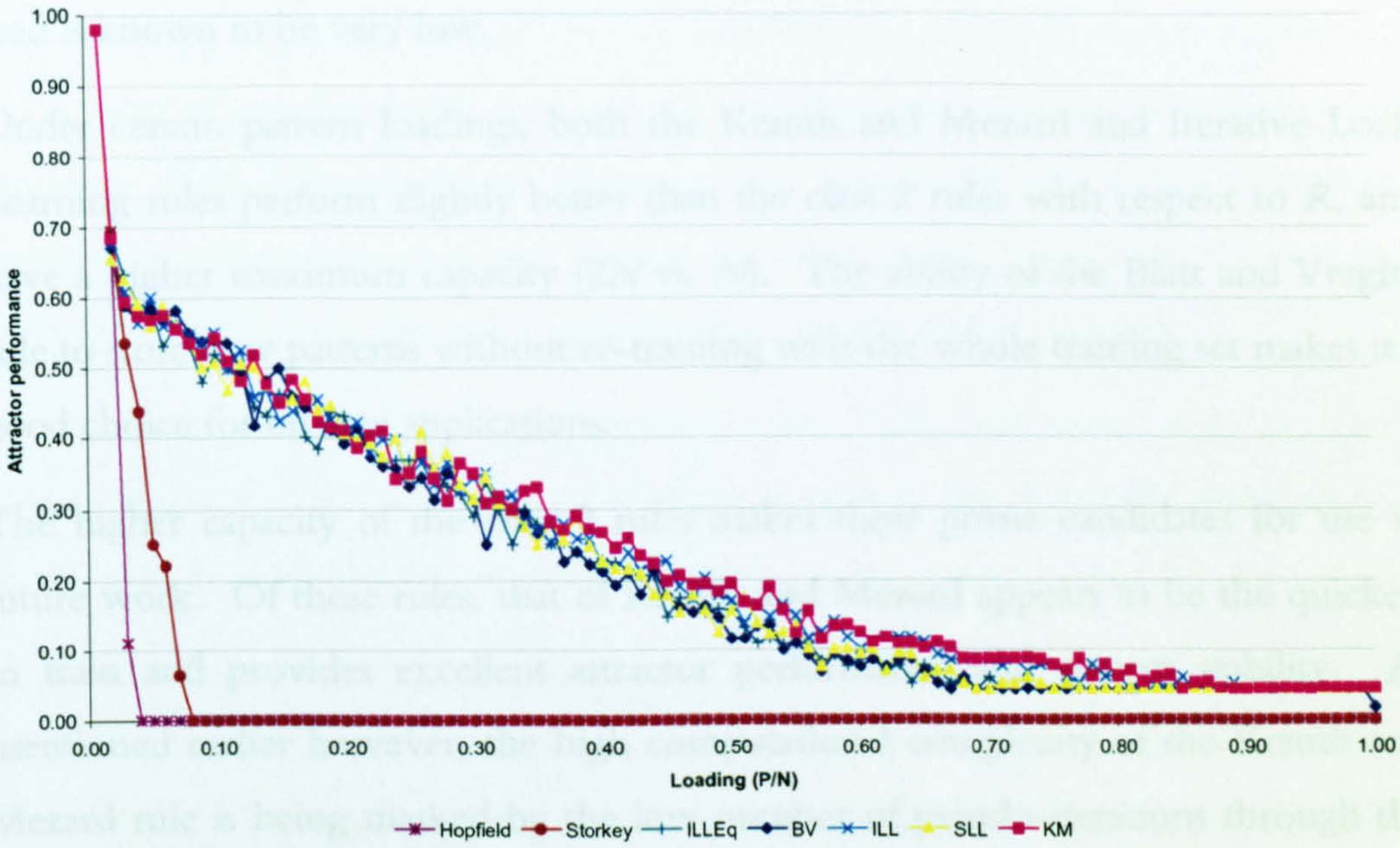


Figure 4.6: Attractor performance of networks learning random patterns of bias 0.8.

As was seen for unbiased random patterns, the performance of all the class 2 and 3 learning rules is very similar. The small difference in performance between the pair of rules KM and ILL and the rest of the high performance algorithms that was seen in figure 4.5 is more evident for biased data. The difference is again only discernable over the loading range α=0.40 (40 patterns) to α=0.80 (80 patterns).

Using the class 2 and 3 learning rules with either unbiased or biased data, a sharp initial fall in attractor performance occurs at low loading. Following that, the attractor performance falls steadily approaching some minimum level. The non-zero minimum value of the attractor performance is likely to be attributable to the fact that, as the values for the attractor performance are averaged over 5 networks, it is eminently possible that some successful convergences will occur for some of the patterns at even high loadings.

4.5. Conclusions and Summary

The results have shown that the class 1 learning rules, Hopfield and Storkey, are not appropriate for either high network loadings or correlated patterns. Both learning rules suffer from poor pattern stability and a rapid decline in attractor performance with increasing pattern loads though the Storkey rule does give better performance than Hopfield. They do have the advantage of being extremely rapid in their ability to train the network by virtue of their being single-shot learning rules. This could make the Storkey rule particularly attractive in situations where the pattern load is known to be very low.

Under certain pattern loadings, both the Krauth and Mezard and Iterative Local Learning rules perform slightly better than the class 2 rules with respect to R, and have a higher maximum capacity ($2N$ vs. N). The ability of the Blatt and Vergini rule to store new patterns without re-training with the whole training set makes it a good choice for on-line applications.

The higher capacity of the class 2 rules makes these prime candidates for use in future work. Of these rules, that of Krauth and Mezard appears to be the quickest to train and provides excellent attractor performance and pattern stability. As mentioned earlier however, the high computational complexity of the Krauth and Mezard rule is being masked by the low number of pseudo-iterations through the training set.

Of the two remaining rules, Iterative Local Learning and Symmetric Local Learning, ILL has the edge in terms of attractor performance and training time especially for the highly biased training patterns. The production of a symmetric weight matrix and thus the guarantee of simple network update dynamics is a not inconsiderable advantage. Given the relatively small difference in attractor performance and the benefit of a symmetric weight matrix, the choice was made to use Symmetric Local Learning for future investigations.

In summary, this chapter has evaluated the performance characteristics of a number of learning rules applicable to the Hopfield architecture. It has been demonstrated that a clear performance gap exists between the class 1, Hopfield-type learning rules and the class 2 and 3, pseudo-inverse approximator and Gardner-type learning rules. Symmetric Local Learning was chosen to be the principle learning rule for the remainder of this investigation.

5. INTRODUCTION TO SPARSE CONNECTIVITY

5.1. Introduction

It was noted early in this work (c.f. chapter 1) that a strong motivation behind this body of research was to try and reduce the cost of implementing Hopfield-type associative memories. The form that cost takes depends on the nature of the implementation being attempted. If the implementation is in software, the cost is usually in CPU time and memory as the storage requirements and computational complexity of the networks can often be considerable and scale exponentially with the number of neurons being used. If, on the other hand, the implementation is in hardware there arise additional physical costs. Implementing a fully-connected network as a computer chip will require a large amount of silicon real-estate and again the complexity scales exponentially with the size of the network.

A potential solution to this problem for both software and hardware implementations is to reduce the level of complexity of the network. Typically this means reducing the number of connections.

As the link between the connections and the weights is an inseparable one and the information stored in networks of the type studied here is contained in the weights it seems clear that any reduction in the level of connectivity will impair the ability of a network to operate at its full potential.

It is hoped however, that a balance may be struck between a network's level of connectivity and its performance. This phase of the work relies on the fact that, in the past, the majority of work analysing the performance of associative memory networks has invariably used only random patterns as test data. It will be shown that the more structured nature of more 'natural' data can aid in circumventing some of the negative aspects of a reduction in network connectivity. In the case of the work undertaken in this project, natural data takes the form of images of man-made constructs, typified by objects of largely the same colour. Doors, cars, buildings, are all examples of such constructs. This work uses artificially generated patterns to simulate these characteristics.

Natural data is in many ways different from random data. It will be demonstrated that these differences should lead to improved performance under certain conditions.

5.2. Justification of Approach

It was seen earlier when analysing the performance of fully-connected networks (c.f. chapter 4) that the Symmetric Local Learning algorithm of Gardner (1988) was a strong performer. Using learning rules of this type allows the individual neurons to be trained as if they were perceptrons.

It is known that the notional maximum capacity of the perceptron is $2N$ patterns for unbiased, random data (Cover, 1965; Gardner, 1988) where N in this case is the number of inputs to the perceptron. Gardner has also shown (c.f. figure 5.6) however, that the capacity of a network of perceptrons increases as the bias of the training set rises. For this to be the case, the capacity of the individual perceptrons must also rise.

Lopez et al. (1995) have shown that the increase in capacity is not solely dependant on a rise in training set bias but is also strongly related to the correlations present within the training patterns. An example illustrates this:

Input pattern						Output value
1	1	1	1	-1	1	1
-1	1	1	1	1	1	1
-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	-1	1	-1

Figure 5.1: A selection of input patterns and the corresponding output values.

Figure 5.1 shows a selection of input patterns and their corresponding output values. It is clear that each individual pattern is highly biased but taken over the entire training set the bias evens out; the number of positive and negative values is the same. Lopez et al. construct an argument showing that when correlated subsets of the input patterns (pairs, triples, etc) also have correlated outputs, then there should be an improvement in capacity. This would indicate the data set shown in figure 5.1 should be more easily learnable than a set of random patterns.

In the work of Lopez et al. only pairwise correlation is initially considered. Pairwise correlation in this instance does not mean every possible pairing of two patterns but rather, for instance, the first pattern with the second and the third pattern with the fourth. This is an obvious simplification of the perceptron's environment but suffices for the purposes of the proof and is later extended to a more general case.

Taking the following training set as an extreme example.

Input pattern						Output value
1	1	1	1	1	1	1
1	1	1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1

Figure 5.2: An example training set consisting of paired duplicate patterns.

It should be noted that the training set illustrated by figure 5.2 consists of paired duplicate patterns. This extreme case gives Lopez et al. the basis for their argument.

A term R is defined to describe the similarity of two patterns as a value indicating positive or negative overlap between the patterns.

$$R = \frac{1}{N}\boldsymbol{\sigma}^{\mu}\boldsymbol{\sigma}^{\nu} \tag{5.1}$$

where N is the size of the input pattern and the $\boldsymbol{\sigma}$'s are the input patterns transformed according to:

$$\boldsymbol{\sigma}^{\mu} = \xi^{\mu} s^{\mu} \tag{5.2}$$

Care must be taken here as the notation differs slightly from that used elsewhere in this work. In equation (5.2), the ξ's are the input portions of the complete patterns and the s's, the outputs. This transformation has the effect of reversing the values of an input pattern in the case where the output is -1. This is done so that patterns which are identical as regards the input but dissimilar in output are represented in a maximally dissimilar way for the purposes of calculating the overlap, R.

Input pattern						Output value
1	-1	1	1	-1	1	1
1	-1	1	1	-1	1	-1

Figure 5.3: An example of a pair of patterns with identical inputs but dissimilar outputs.

Figure 5.3 (above) shows two example patterns with which this may be illustrated. It should be clear that two patterns with identical inputs cannot be classified into more than one class by a perceptron. Calculating R for the pair of input patterns as they currently are would give a value of 1, indicating that they were the same. Transforming the patterns as per equation (5.2) provides a useful means of representing pairs of patterns which are unlearnable.

The transformed patterns look as follows:

Input pattern					
1	-1	1	1	-1	1
-1	1	-1	-1	1	-1

Figure 5.4: Transformed form of the input patterns shown in figure 5.3.

Using the transformed patterns shown in figure 5.4, calculating R now gives us a value of -1 indicating the problem that the patterns have the same output value without having to further involve the output value in the calculation.

So, to summarise the effect of calculating R for a pair of patterns: a value of R equal to 0 indicates that there is no correlation between the pair of patterns. A value of 1 occurs when the patterns are identical. A value of -1 indicates that the input portions of the patterns are identical but the outputs are different and so the pair is unlearnable.

R is calculated over all pairs of patterns and is therefore a measure of the mean pairwise overlap.

Having established exactly how the overlap is calculated it is now possible to prove the critical capacity α_c for various values of R.

For R=0, the patterns are uncorrelated and $\alpha_c(R=0)=2$, as per the results of Cover (1965) and Gardner (1988).

For R=1, it should be apparent that as the patterns within the pairs are identical storing the first pattern of a pair implies the storage of the second. Therefore, in this case the capacity is doubled and $\alpha_c(R=1)=4$.

For R=-1, the implication is that all pairs of patterns are linearly inseparable and so the very first pair must also be so. This first pair renders the rest of the patterns unlearnable and in this instance $\alpha_c(R=-1)=0$.

As Lopez et al.'s argument takes place under the condition of $N\rightarrow\infty$, for values of R even very close to -1 the linear separability of the patterns can be guaranteed.

The relationship between R and α_c is shown as a graph (figure 5.5) of values produced by experimental means as part of the work of Lopez et al.

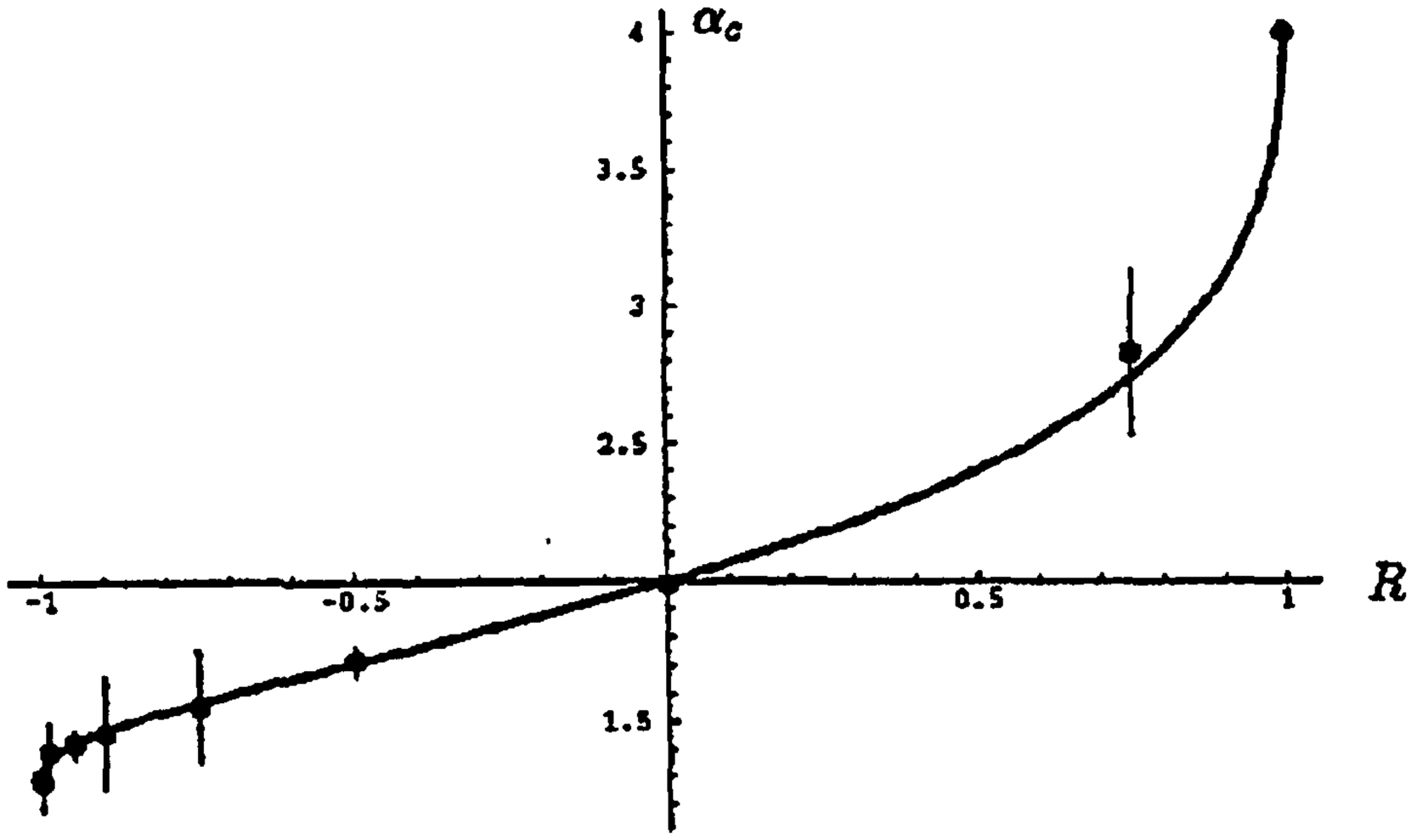


Figure 5.5: Graph showing the relationship between the mean pattern overlap, R, and the critical capacity α_c.

Lopez et. al. go on to produce a proof that is not restricted to simple pattern pairs but involves pairwise correlation between arbitrarily sized m-tuples. The implication of this is that if a high degree of correlation exists between patterns within a tuple that is sufficiently large, i.e. at or near the size of the set of input patterns then the capacity of the perceptron should be much higher than the standard $2N$.

Considering again a network of perceptrons, Gardner (1988) showed that as the critical capacity (α_c) of a network improves so does the value of the smallest stability coefficient (c.f. §2.4.1), K, as the correlation between the patterns increases. This is illustrated by the following graph (figure 5.6) reproduced from Gardner's work.

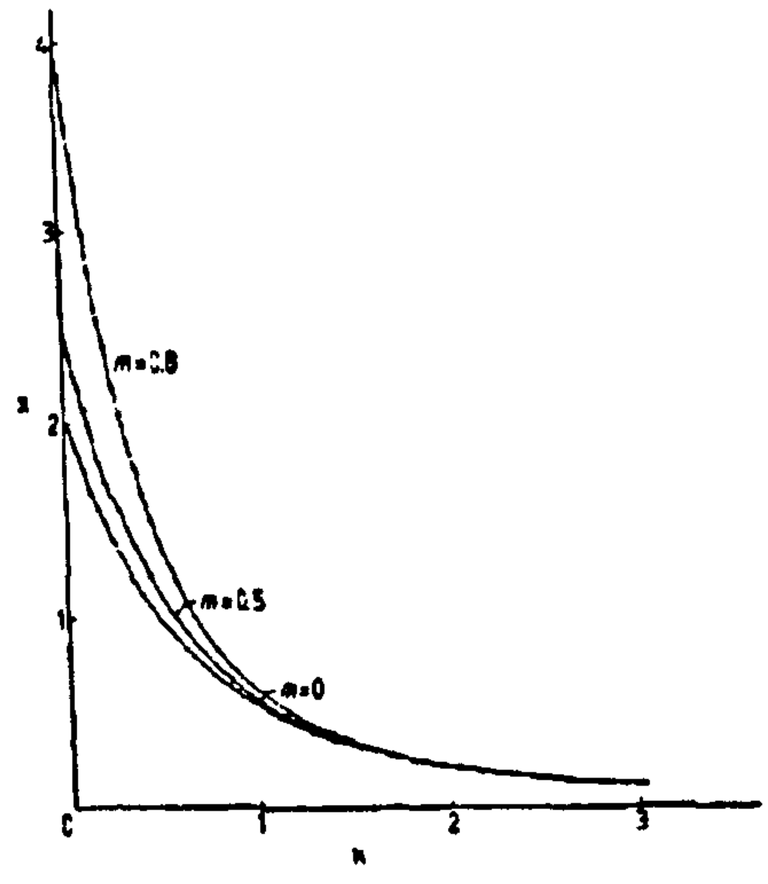


Figure 5.6: Graph showing the relationship between the critical capacity α_c (y-axis) and the minimum stability coefficient, K (x-axis), at increasing levels of pattern correlation indicated by the magnetism of the patterns, m.

It can be seen from figure 5.6 (previous page) that as the correlation between the patterns rises (indicated by the increasing magnetism, *m*) from 0 to 0.8, the value of K at equivalent α_c also increases. The magnetism, though analogous, differs slightly from the traditional measure of pattern bias in that a value of 0 indicates unbiased patterns and rises to 1 for fully biased patterns.

The magnitude of K is important as it has been shown that larger values of K should imply larger basins of attraction (Gardner, 1988; Kepler and Abbot, 1988).

In summary, when correlated patterns share the same output value it can be expected that significantly better performance for perceptrons both in terms of capacity and attractor performance will arise. The key idea explored here is that the connectivity pattern adopted in a dilute network effectively defines a new training set.

The results of performance analyses on networks learning random data at the same level of bias as the locally correlated data will provide an indication of whether or not locally correlated data does indeed lend itself favourably to local connection topologies.

5.3. Review of Literature Related to Sparse Connectivity

Given the unrealistic assumption of full connectivity between neurons it makes sense to pose the question as to the effectiveness of Hopfield-type networks within which the level of connectivity has been reduced. There are three distinct ways in which this may be achieved:

- Training networks using sparse *binary* patterns (i.e. patterns with few 1's present). The manner in which this reduces the level of connectivity is discussed in more detail below.

- Eliminating connections from a network which has already successfully learnt a set of training patterns. This will be termed *post-training dilution.*

- Training networks in which sparse connectivity has already been established through some strategy or heuristic. This will be termed *sparse connectivity.*

As this work is concerned with simple dynamics, the results of research on networks with asymmetric connectivity are not considered in any great detail. The presence of such work is acknowledged however and is briefly summarised alongside its symmetrical dilution counterparts.

Training Using Sparse Binary Patterns

Training networks using sparse binary patterns is a common enough technique in the field of associative memories though it is not immediately clear why this should have the effect of reducing the level of connectivity of a network. Training a network using a set of very sparse binary patterns will result in a weight matrix with a large number of zero-valued weights. These weights therefore play no part in the network's update dynamics and the connections are effectively non-existent.

An example of training in this manner can be seen in the work of Levy et al. (1999) in which sparse patterns are used in conjunction with a multi-modular network to study the effect of storing individual patterns in varying numbers of modules. The number of modules a pattern occupies is termed its *coding level* and they show that patterns with larger coding levels are more resilient to intra-module synaptic damage. The network is trained using a modified version of the Hebb-style learning rule developed by Tsodyks (1989).

Although of passing interest, Levy et al.'s work is biologically motivated with the aim of modelling simple cortical function. Due to the neurobiological bent and the use of binary patterns (as opposed to bipolar) to create low effective connectivity this technique is not considered further in this work.

Diluting Networks Trained using One-Shot Hebbian Learning

The problem of diluting a network trained using Hebbian learning is exactly equivalent to that of training a previously diluted network with the same rule. This fact arises due to the independence of each neural bond during training. The presence, or otherwise, of a particular bond has no bearing on the training of any other.

One of the earliest examples of the investigation of the effect that dilution of connectivity might have on Hopfield-type networks is that of Sompolinsky (1986). In this work, synapses are removed from the network symmetrically (i.e. W_{ij} & W_{ji}) and the pairs are chosen at random. The capacity of the network, trained using the Hebb rule, was found to fall almost linearly with d, the proportion of connections removed. The quality of the recalled patterns, as a function of the overlap of the post-retrieval network state with the original pattern, falls far less sharply indicating that the level of connectivity in this case has less bearing on the pattern stability than on the critical capacity of the network.

Sompolinksy's work appears to have initiated a burst of activity in the area of diluted associative memory models. The next major contribution was that of Derrida et al. (1987) in which an asymmetrically diluted model was studied from a analytical viewpoint with the aim of better understanding the dynamics of such an architecture. Derrida's analysis continued with a further work (Derrida, 1989) in which the distribution of neural activities for the stored patterns was examined.

The effect of random dilution on networks trained using random biased patterns was examined by Evans (1989) using networks trained using the modified one-shot Hebb-type rule developed by Tsodyks and Feigel'man (1988). Evans notes that dilution does not result in stored patterns gradually declining in performance as attractors but rather more complex mechanisms occur with the possibility of a memory becoming a limit cycle. The potential for this was mentioned previously (c.f. chapter 2) as a consequence of asymmetric connectivity.

da Silva et al. (1995) study the generalisation capability of an extreme and asymmetrically diluted version of the Hopfield model. Generalisation is the ability to group a given set of correlated patterns into distinct classes. They show that dilution improves the performance of the network as a categorisation device compared with the fully-connected Hopfield model. It is stated within

da Silva et al. that Derrida et al. (1987) proved that an asymmetric, diluted version of the Hopfield model could not only recognise the patterns which had been stored but also had greater capacity. Building on that work, da Silva et al. study this model further and show that dilution and asymmetry also improve the generalisation ability of the model. They conclude that their model is more biologically realistic, a fact somewhat justified in that they employ both dilution and asymmetric connectivity. The generalisation improvement comes in the form of requiring fewer example patterns in order to be able to classify the input patterns correctly.

An important and oft-cited work was produced by Canning and Gardner (1988) examining symmetrically dilute models of neural networks trained using the Hebb rule. The focus of this work is on more structured topologies motivated by both the realisation that fully-connected systems or those possessing long-range connections would be difficult to build physically and that beneficial correlations in 'real' problems are likely to be local. Some biological motivation is present in that the topology inherent in a 3-dimensional neural system is acknowledged and Gardner-Medwin (1976) is cited as discussing the links between the brain and recurrent networks. Canning and Gardner show that random connectivity is the best choice of connection architecture for maximising the ratio of the number of patterns stored to the input dimensionality of each neuron, a measure which is termed *storage efficiency* within this work and was described in chapter 4.

Komoda et al. (1991) examine the way in which the robustness of the stored patterns fares against random dilution using an already dilute network. The areas of performance investigated were the attractor overlap or similarity of the retrieved states with the stored ones, attractor basin size, and storage capacity. It is noted that the networks used deteriorate on all three counts with increasing dilution and that disruption is worst in networks with small aligned local fields (c.f. chapter 2). Komoda et al. show patterns stored in a network employing a Gardner-class high capacity learning rule have a strong robustness at low levels of dilution while for networks trained using the Hopfield rule the opposite is true, performance is better at higher levels of dilution. Their result should perhaps not be surprising as the capacities of these two architectures are very different and it is likely that in the case of a network trained to full capacity with a Garder-class rule that any significant

degree of perturbation in the weights will lead to recall and capacity degradation as is shown by the work of Bouten et al. (1990), summarised in the next section.

The symmetrically dilute Hopfield network was also examined with regard to the network's ability to act as a categorisation device by Krebs and Theumann (1999). It is claimed that the categorisation performance is enhanced by the dilution and exceeds that of the fully-connected model. This work is not dissimilar to that of da Silva et al. (1995), mentioned above in the context of asymmetric dilution.

Post-Training Dilution of Connectivity

A considerable amount of the work on the subject of post-training dilution has a biological focus. Analogies have often been drawn between neurological disorders such as Alzheimer's disease and the effect of the removal of synapses from artificial neural networks. These biologically motivated approaches are of interest but differ significantly from this work in their goals and are therefore only briefly summarised below.

The work of Ruppin and Reggia (1995) falls into this category. In their work they present an 'analytical framework' for estimating the functional damage arising from the removal of connections in a structured manner using a network trained using sparse binary patterns.

Chechik et al. (1998) are similarly biologically motivated but follow a developmental approach, looking at the manner in which the young brain exhibits synaptic overgrowth followed by selective reduction of synapses. The reasoning behind this is that neural connectivity is expensive in terms of energy and pruning is one means by which the body seeks to reduce energy consumption. Chechik et al. present several strategies for diluting synapses and show a link between the results of their work and certain types of amnesia.

An extensive review of the range of work being conducted in this area can be found in (Ruppin, 1995; Ruppin and Reggia, 1998).

Non-biologically motivated work on post-training dilution of synapses is far less widespread. Prior work on the post-training dilution of networks trained using non-Hebb-type rules is summarised below.

Vishwanathan (1995) studied the fault tolerance of neuronal failure using networks of perceptrons by examining the proportion of patterns that continue to be recalled

without error when some of the neurons fail. This work is continued in (Vishwanathan, 1995) in which the effect of removing synapses of particular magnitudes on the recall performance of similar networks is determined by mathematical means. It is shown that a best-case upper bound on the amount of retrieval error introduced through removal of synapses can be estimated using statistical mechanics.

Sparse Connectivity

Bouten (1990) examines two strategies for establishing sparse connectivity in networks using a high capacity learning rule. The first of these is a simple random removal process which, it is shown, leads to a linear dependence on the proportion of connections being removed. This result corresponds to that of Sompolinsky (1986) who used the Hebb rule to store the patterns in the network. The second strategy, termed *annealed dilution*, chooses the synapses to eliminate based upon the nature of the training set, contributing to the storage of the patterns. This form of dilution is functionally equivalent to training a fully-connected network and removing weights in order of ascending absolute magnitude.

Annealed dilution is shown to provide a significant capacity improvement over random dilution though it should be noted that the resulting architecture is tailored very specifically to the actual data being learnt. Bouten's work is theoretical and no empirical results are presented.

Stiefvater et al. (1993) propose a sparsely-connected Hopfield-type network for recognising natural, highly correlated data in the form of video images. The training data has a high level of both inter-pattern correlation and site correlation, terms that will be explained in detail later in this work. Their studies have shown that, due to unfavourable correlations in the training data, models originally proposed for the processing of correlated random (biased) patterns fail to work on the 'natural' data. The spatial and temporal continuity of nature causes inter-pattern and site correlation to be common features of data derived from real-world sources. The network appears to operate on quite heavily pre-processed images. Video images have applied to them a 'Gabor filter' designed to mimic the functionality of simple cells in the visual cortex.

Stiefvater et al. present the notion of 'practical usability'. In order for this to be the case for a given network they state that network relaxation (recall) times

should be of the order of a few seconds and that learning times should be reasonable. Though a definition of 'reasonable' is not given it would be unreasonable to assume minutes or hours rather than days.

Stiefvater et al. note Canning and Gardner's (1988) work showing that diluted network models are efficient at processing random patterns and the effectiveness of Bouten et al.'s (1990) 'metastabilization' technique using learning and dilution. It is suggested that these techniques might be just as applicable to training patterns with a natural correlation structure as the dilution of the network could be tailored to match. It was demonstrated that, for random patterns, annealed dilution strategies produced networks with larger basins of attraction than might be found in networks where the structure forms uniform geometric neighbourhoods. Justification for the presence of this phenomenon is given. It is stated that in the case of local neighbourhood connectivity, important long-range interactions are cut. Quite why these interactions are important is never discussed. The question arising from this is whether or not the high-valued synapses that would be chosen to be kept during the process of annealed dilution are the same as those that would form local neighbourhood connectivity in the case of naturally-derived training data. If this turned out to be the case then networks created in this way would begin to correspond more closely to their biological counterparts since, according to Mallot and von Seelen (1989), computation by uniformly structured connections appears to be an important factor in neural information processing. A local network topology would also be easier to implement in hardware due to the reduced physical 'real estate' requirements.

Heuristics were devised and developed by Stiefvater et al. (1993) which would create connectivity patterns based on statistical analyses of the training data and it was shown to be the case that a local neighbourhood connectivity topology does indeed select the highest valued synapses as would occur using the annealing technique. Within their work, three novel learning techniques are considered: a geometric one, a system dependent on site statistics, and a combination strategy. The geometric technique seeks simply to define a regular neighbourhood of connectivity for all neurons. The site statistics method attempts to cut connections based on the level of cross-pattern activity at each neuron. Neurons

with activities close to the mean level of activity are deemed to be the most 'important' and are kept.

Neighbourhood connection strategies were also proposed by Karholm (1993) in his work on associative memories with short-range, *higher order* couplings. The higher order couplings are capable of computing the product of neuron inputs and, it is stated, appear in the brain.

Karlholm illustrates the problem of linear inseparability of patterns with regard to using a local neighbourhood of connections arguing that: "*...if the range of connections is restricted to a small neighbourhood, it may happen quite often that patterns look the same from a single unit's point of view*". It is argued that a neighbourhood size should be sought that minimises the conflict between the training patterns. This is the same argument as was used earlier for the expectation of improved performance in the networks used in this work (c.f. §5.2).

Architectures modelling hierarchical connection topologies have been quite popular areas for study. The majority of this research takes as its inspiration the work of Marr (1971) and his theory of the function of the mammalian archicortex as a memory.

Sutton et al. (1988) propose a hierarchical model of memory based on the principle that regions of the cortex are topographically organised into nested subnetworks. The hierarchy has three levels, the first taking the form of a number of individually fully-connected but separate Hopfield-type networks. These *first-level clusters* are linked by a subset of connections termed *projection elements* to form *second-level clusters*. Further connections link second-level clusters together to form *third-level clusters*.

The technique of Sutton et al. can be continued to establish as deep a network hierarchy as is required. The motivation behind the work is to examine not just the storage capability of the model but also to develop the model as a tool that may be useful in modelling memory loss in neurodegenerative disorders such as Alzheimer's disease. The training and update dynamics of this network are complex and beyond the scope of this work but Sutton et al. show experimentally the results on memory recall of various degrees of dilution of inter-cluster connectivity.

O'Kane and Treves (1992) take a similar approach to that of Sutton et al. although they only consider two levels in their hierarchy. Fully-connected networks, termed *modules*, are joined with each other using a subset of connections distributed at random. Patterns are stored on both the short- and long-range connections using Hebb-style rules. The attractor states of the network and the storage capacity are examined using the statistical physics techniques made popular in this field by Derrida et al. (1987).

O'Kane and Treves conclude that their network is not a viable model for the organisation of memory in the cortex. They reason that the storage capacity of a neural network scaling with the number of connections per unit rather than with the size of the system is 'wholly implausible from a biological point of view'.

A comprehensive review of the field of modular neural networks can be found in (Caelli, Guan et al., 1999) in which the authors note the absence of work involving the incorporation of geometric structure into neural models.

Jacobs and Jordan (1992) present an examination of the computational consequences of a bias towards short connections in neural networks. While their work is not restricted to networks of the Hopfield type they present some interesting thoughts on the justification and motivation for such topologies primarily related to the speed of electrical signal propagation in biological neural connections. Jacobs and Jordan cite evidence suggesting that cognitive processes are less localised in newborns than in adults (O'Leary, 1989; Greenfield, 1991). This notion of radical topological change during human development was also central to the work of Chechik et al. (1998), mentioned earlier.

5.4. Summary of Literature Review

To summarise the important works relevant to this investigation:

Canning and Gardner (1988) mention the possibility that beneficial correlations are likely to be local in natural patterns and that benefits might be gained by creating neighbourhood connectivity at the same distance.

Karlholm (1993) reinforces the hypothesis of Canning and Gardner by stating explicitly that patterns that appear similar from an individual neuron's perspective might arise by restricting connectivity to a small neighbourhood around each neuron.

Lopez et al. (1995) demonstrate that, for perceptrons, learning patterns with correlated inputs and identical outputs will lead to improved capacity.

It is known from the work of Gardner (1988) that higher capacity in a network of perceptrons leads to an increase in the minimum value of the stability coefficients.

Finally, it has been shown that larger stability coefficients imply larger attractor basins and so better attractor performance should result (Gardner, 1988; Kepler and Abbot, 1988).

6. POST-TRAINING REMOVAL OF SYNAPSES AND ITS EFFECT ON NETWORK PERFORMANCE

6.1. Overview

In the search for an efficient, reduced level of network connectivity it makes a certain amount of sense to examine the effect of removing connections from a network that has previously been trained using one or more of the algorithms known to be applicable to fully-connected architectures. Success in training networks this way could provide a means whereby training could be performed off-line, in software, and the weights might then be transferred to a hardware-based network.

While strategies of this nature are going to be unable to deliver any reduction in the training time of the networks (indeed, the overhead of subsequently removing connections from a trained network contributes to an increase in overall network preparation time) there will be savings in terms of the amount of memory required to store the remaining connections and weights and a corresponding reduction in the hardware costs of any physical realisation of such networks but only after dilution has taken place.

Prior work in this area appears to have been largely restricted to either examination of networks trained using Hebbian learning only or random dilution strategies (Sompolinsky, 1986; Kothari and Lotlikar, 1997). It has been shown that the capacity of such networks falls linearly with the proportion of connections removed using such strategies (Sompolinsky, 1986). As justification for treating trained networks in this manner, Chechik et al. (1998) pursued the biologically motivated notion that synaptic pruning during the development of the mammalian brain was an attempt by the brain itself to reduce the energy requirements of a system which, when immature, was both infrastructure overloaded and energy inefficient. This hypothesis seems appropriate to the creation of physical artificial networks also, especially with respect to infrastructure complexity. A review of work in this area was presented in chapter 5.

The experiments reported here use networks trained on random data using one of two high-performance learning rules. The trained networks are analysed with respect to the stability of the patterns being learned, and the ability of the

networks to recall original patterns from a corrupt example (attractor performance).

6.2. Experimental Design

The networks used in this series of experiments were 100 neurons in size and the units were fully inter-connected. The weight matrices were generated using either the Symmetric Local Learning algorithm (Gardner, Gutfreund et al., 1989) or the Blatt and Vergini (1991) method for approximating the pseudo-inverse. These learning rules were covered in detail in chapter 2.

Randomly generated training data at two levels of bias (0.5 and 0.9) were used. The choices of biases were made so that one provides unbiased patterns and the other patterns that are biased heavily towards +1 values.

The trained networks are analysed in two ways: The first analysis is that of pattern stability. A pattern is stable if, when applied to the network as a start state, the network state does not change after all neurons have been updated. The proportion of patterns stable at each network loading is reported for various degrees of synaptic removal.

The second analysis is that of attractor performance. This analysis uses the modified Kanter and Sompolinsky (1987) measure as detailed in chapter 4. The purpose of this analysis is to find out whether or not attractor performance decreases gracefully with increasing dilution or a sharper change in performance occurs.

6.3. Synapse Removal Strategies

Two synapse removal strategies were employed in this series of experiments: random removal and smallest-value-first removal.

6.3.1. Random Removal

In the case of random synapse removal a value for the proportion of connections to be removed is chosen. This value is multiplied by the number of connections within the fully-connected network and then halved to give the number of connection pairs to be removed. Then, a pair of units is chosen at random and, if a connection between the pair exists, the bi-directional link is removed. This is repeated until the desired level of connectivity is achieved. Ensuring that the bi-directional link is fully removed maintains symmetry within the weight matrix thus ensuring simple update dynamics (c.f. chapter 2).

6.3.2. Smallest-Value-First Removal

The number of connection pairs to be removed is determined in the same way as for random removal. The network's connections are then scanned to find the weight with the smallest absolute value (that which is closest to zero). Once the connection with the smallest weight value has been identified it is removed. The process continues until the required number of connections has been eliminated.

A functionally equivalent strategy was proposed by Bouten (1990) and named *annealed dilution*. Bouten presented an analysis of a theoretical network in which a number of weights were omitted. The absent weights are the same as those that would be removed were smallest-value-first removal performed on a trained fully-connected network.

For brevity, this scheme is henceforth referred to as smallest-first removal.

6.4. Results

This section presents the results of the experiments outlined in §6.2. The results are initially categorised by learning rule and subsequently by analysis type.

6.4.1. Symmetric Local Learning

6.4.1.1. Pattern Stability

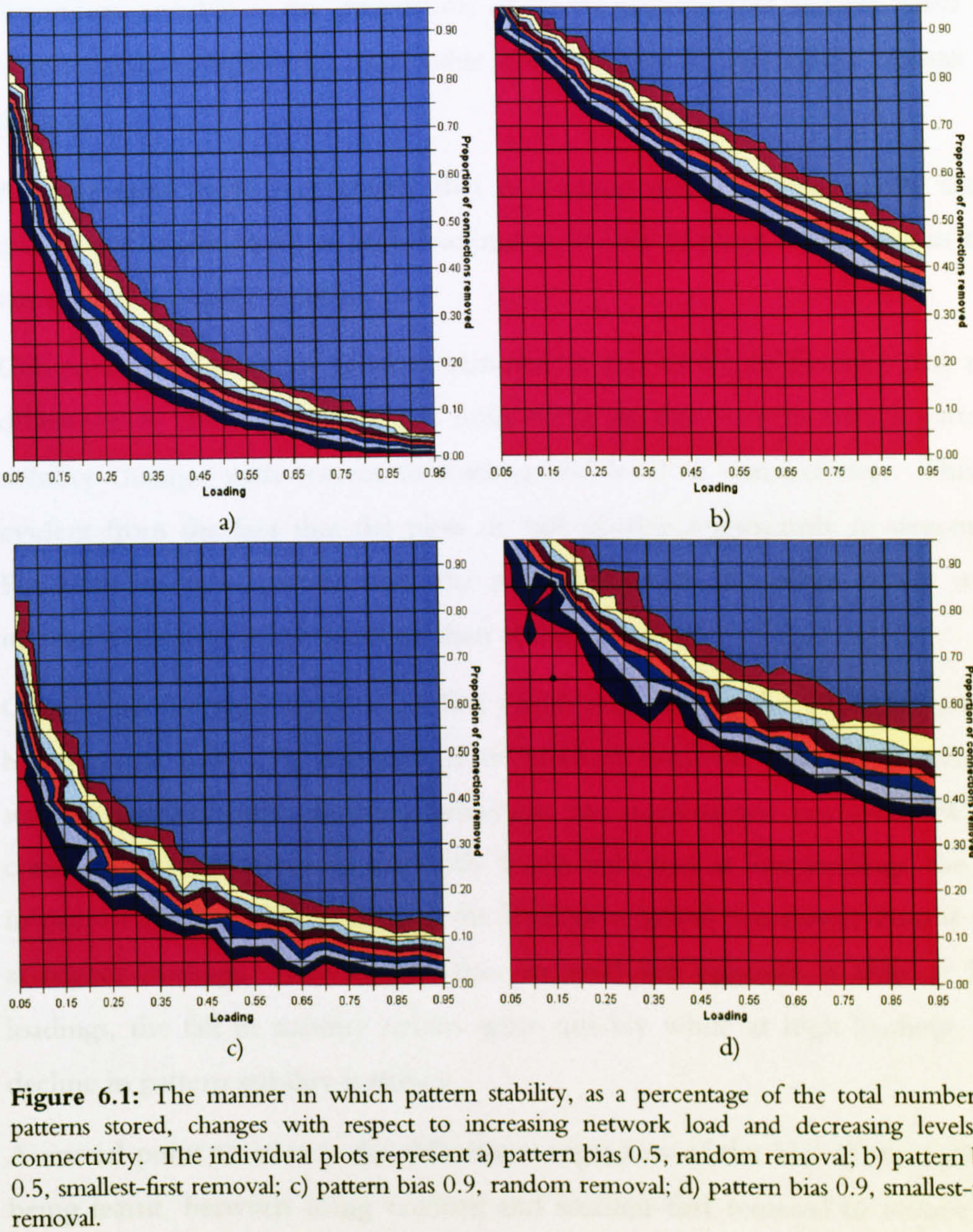


Figure 6.1: The manner in which pattern stability, as a percentage of the total number of patterns stored, changes with respect to increasing network load and decreasing levels of connectivity. The individual plots represent a) pattern bias 0.5, random removal; b) pattern bias 0.5, smallest-first removal; c) pattern bias 0.9, random removal; d) pattern bias 0.9, smallest-first removal.

Key: Percentage of patterns stable represented by each plot colour.

0.00-10.00	10.00-20.00	20.00-30.00	30.00-40.00	40.00-50.00
50.00-60.00	60.00-70.00	70.00-80.00	80.00-90.00	90.00-100.00

Figure 6.1 (previous page) shows the following: The plot pairings a-b and c-d are for patterns of bias 0.5 and 0.9 respectively. The left-hand plots in each pairing represent networks in which random removal was performed while the right-hand plots represent those in which smallest-first removal was used.

The plots are 2-dimensional contour maps of a 3-dimensional surface. The dependent variable is the proportion of learnt patterns that remain stable as connections are removed. Each value is represented by one of the colours in the key.

All four plots show very clearly that it is not possible to remove the same quantity of connections at high loadings as at low loadings and maintain the same level of pattern stability.

Comparing the plots for random removal (a and c) it can be seen that the difference in the pattern bias has little effect on the way in which pattern stability changes with respect to loading and level of connectivity. This is evident from the fact that the plots do not change significantly in structure. For both levels of pattern bias, the pattern stability falls more slowly with increased removal at low loadings than at high loadings.

Comparing the plots for smallest-first removal (b and d) the effect of pattern bias remains low. In comparison to random removal, the fall in pattern stability exhibits quite different behaviour for smallest-first removal. When connections were removed randomly it was seen that at low loadings, the fall in pattern stability with respect to the level of removal was slower than it was at higher loadings. For smallest-first removal the opposite is true; at low loadings, the fall in stability occurs quite quickly while at high loadings, the decline in pattern stability is slower.

A crucial point to note is the difference, regardless of the bias of the patterns being learnt, between using random and smallest-first removal to reduce the connectivity of the network. Comparing the left- and right-hand plots in the a-b and c-d pairings it can clearly be seen that the light purple areas representing pattern stability in the 90-100% range are much enlarged in the case of the right-hand plots which depict the results from the networks affected by smallest-first removal.

6.4.1.2. *Attractor Performance*

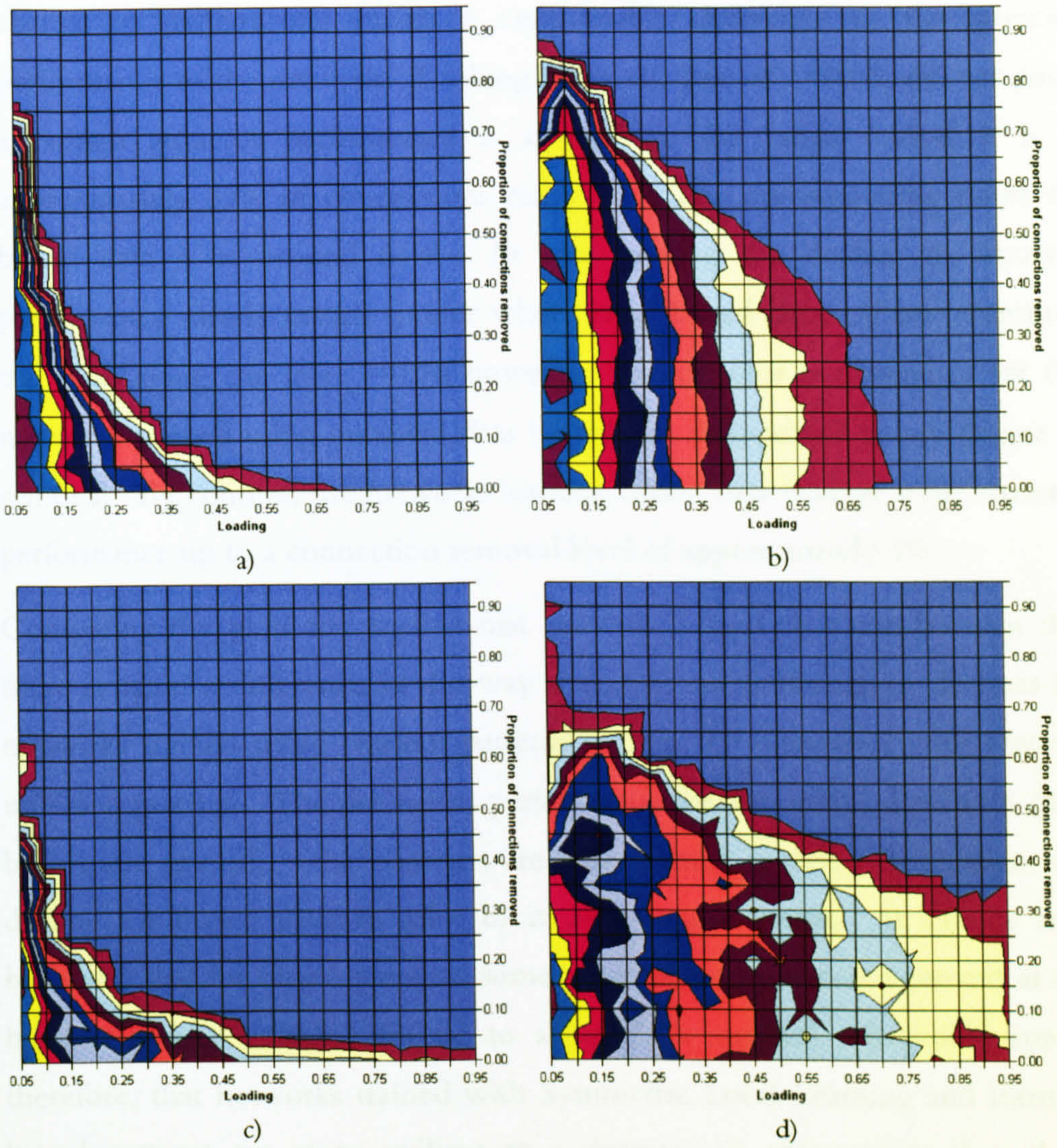


Figure 6.2: The manner in which attractor performance changes with respect to increasing network load and a decreasing level of connectivity. The individual plots represent a) pattern bias 0.5, random removal; b) pattern bias 0.5, smallest-first removal; c) pattern bias 0.9, random removal; d) pattern bias 0.9, smallest-first removal.

Key: Level of attractor performance represented by each plot colour.

0.00-0.05	0.05-0.10	0.10-0.15	0.15-0.20	0.20-0.25	0.25-0.30	0.30-0.35
0.35-0.40	0.40-0.45	0.45-0.50	0.50-0.55	0.55-0.60	0.60-0.65	0.65-0.70
0.70-0.75	0.75-0.80	0.80-0.85	0.85-0.90	0.90-0.95	0.95-1.00	

Figure 6.2 shows the following: The plot pairings a-b and c-d are as for pattern stability. The left-hand plots in each pairing again represent networks in which random dilution was performed while the right-hand plots represent those in which smallest-first removal was used.

While presented in the same way as were the plots of pattern stability, the dependant variable in this case is the attractor performance value as reported by the modified Kanter and Somplinksy measure.

Comparing the plots for random removal (a and c) it can be seen that the change in pattern bias makes a considerable difference to the attractor performance of the network. Looking along the line of =0.05, the last point at which attractor performance is at least in the range 0.05-0.10 is at approximately 82% of connections removed. The corresponding point for biased data is at around 64%. At low loading the decline in attractor performance clearly occurs sooner when using biased data. If one examines the plots along the line =0.70 however, the attractor performance for the networks trained using unbiased data has all but disappeared at any degree of removal. By contrast, the networks learning biased data manage some attractor performance up to a connection removal level of approximately 4%.

Comparing the plots for smallest-first removal (b and d) it can be seen that there is again a difference in the way that attractor performance declines for networks trained using random patterns of bias 0.9 compared with that for unbiased patterns. The decline in performance for the unbiased patterns (plot b) is quite smooth – the contours are fairly evenly spaced. In contrast, the decline for biased patterns (plot b) is uneven and erratic. It can be seen however, that for biased patterns, some attractor performance is present at the highest loading (=0.95) for up to about 35% removal. It would appear therefore, that networks trained with Symmetric Local Learning and learning biased patterns are more resilient to a decrease in connectivity than those learning unbiased patterns.

The improvement in attractor performance that can be gained by reducing connectivity using smallest-first removal can be illustrated by graphing the attractor performance against the proportion of connections removed for specific loadings. Figures 7.3 and 7.4 (next page) demonstrate this for patterns of bias 0.5 and 0.9 respectively. Each graph shows, for loadings of α=0.05, 0.30, and 0.50, the attractor performance, R, of a series of networks against an decreasing level of connectivity. Solid lines represent networks affected by random removal of connectivity while dashed lines with the same symbol represent networks where the connectivity was removed using the smallest-first method. The graphs represent vertical slices, along lines of constant pattern load, of the 3-dimensional surface from which the earlier contour plots were created.

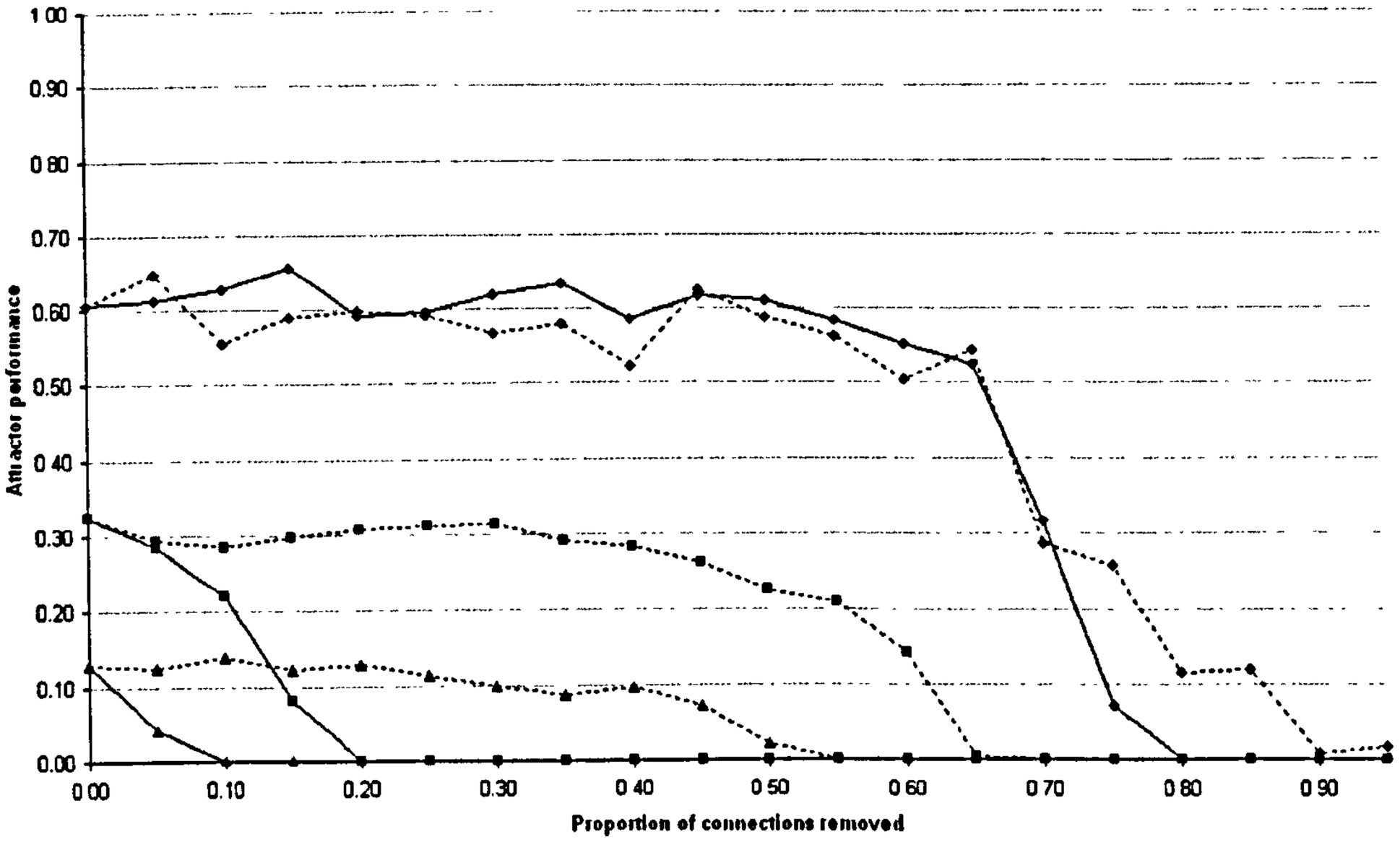


Figure 6.3: The decline in attractor performance for a number of fixed loading points (α=0.05, α=0.30, and α=0.50) using patterns of bias 0.5. The results of both random removal and smallest-first removal are shown for comparison.

Key: Definitions of line styles representing levels of pattern load and type of removal strategy

—◆— 5 patterns, random removal	---◆--- 5 patterns, smallest-first removal
—■— 30 patterns, random removal	---■--- 30 patterns, smallest-first removal
—▲— 50 patterns, random removal	---▲--- 50 patterns, smallest-first removal

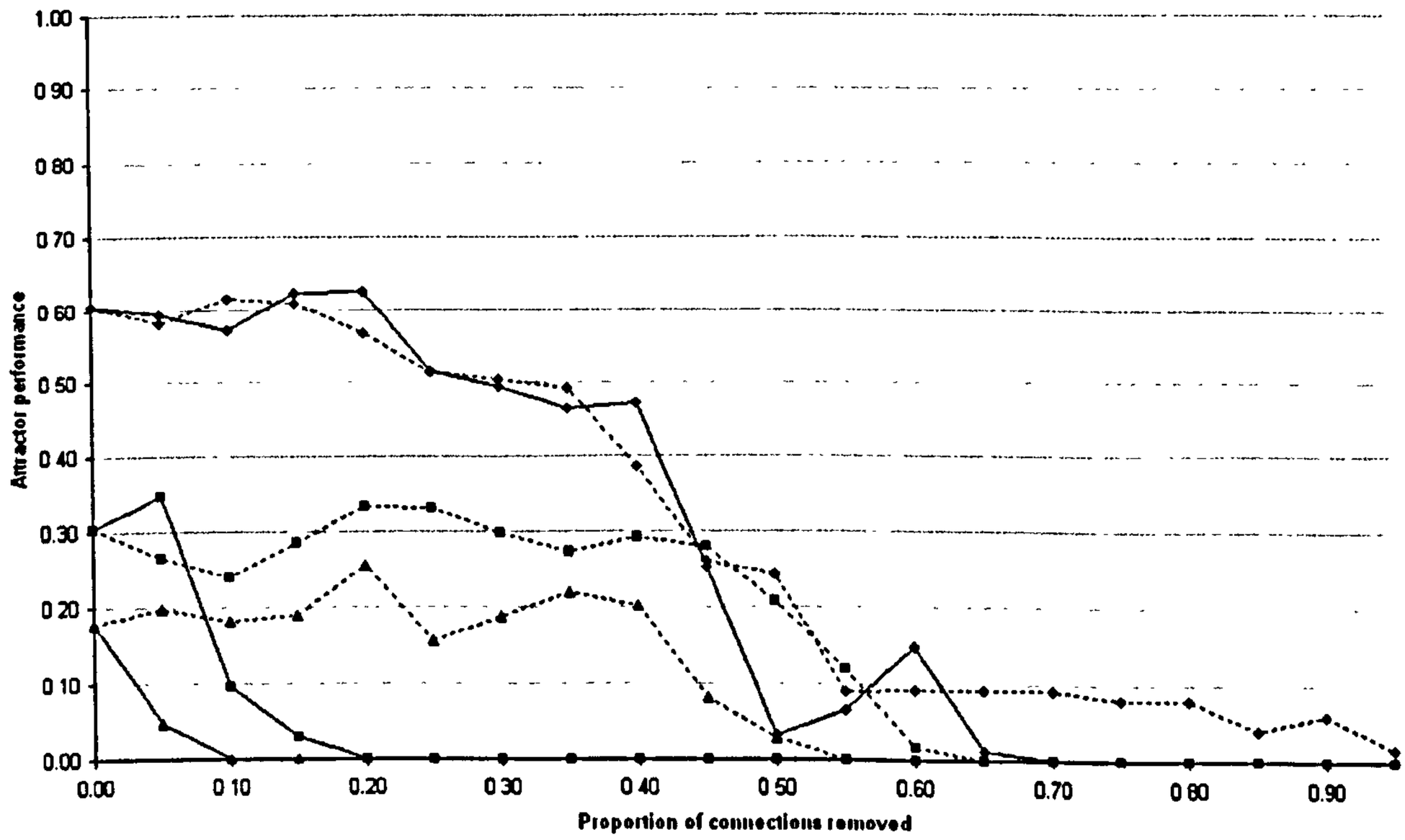


Figure 6.4: The decline in attractor performance for a number of fixed loading points (α=0.05, α=0.30, and α=0.50) using patterns of bias 0.9. The results of both random removal and smallest-first removal are shown for comparison.

Key: Definitions of line styles representing levels of pattern load and type of removal strategy

—◆— 5 patterns, random removal	---◆--- 5 patterns, smallest-first removal
—■— 30 patterns, random removal	---■--- 30 patterns, smallest-first removal
—▲— 50 patterns, random removal	---▲--- 50 patterns, smallest-first removal

Both graphs show that the strategy used for removing connections makes little difference when the network is at low loading (α=0.05) for up to 70% removal in the case of unbiased patterns and up to approximately 60% removal for patterns of bias 0.9. Beyond these levels smallest-first removal begins to show an advantage indicated by the corresponding dashed line being higher than the solid.

For the remaining loadings (α=0.30, and 0.50) the advantage of using the smallest-first removal strategy is more obvious. At even a very small level of connection removal the dashed lines in each of the solid/dashed pairings remain well above their solid counterparts.

6.4.2. Blatt and Vergini

6.4.2.1. Pattern Stability

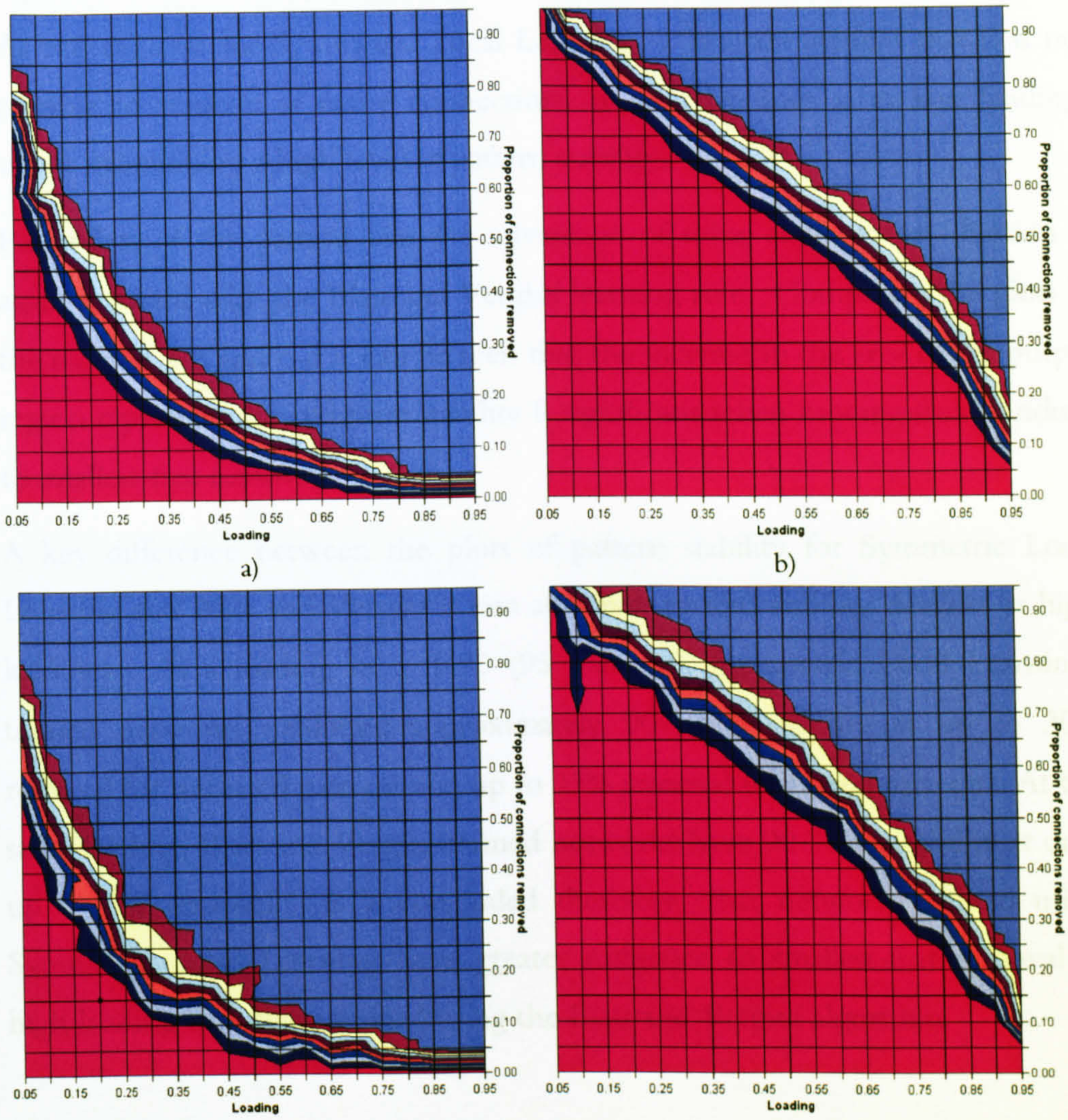


Figure 6.5: The manner in which pattern stability, as a percentage of the total patterns stored, changes with respect to increasing network load and decreasing levels of connectivity. The individual plots represent a) pattern bias 0.5, random removal; b) pattern bias 0.5, smallest-first removal; c) pattern bias 0.9, random removal; d) pattern bias 0.9, smallest-first removal.

Key: Percentage of patterns stable represented by each plot colour.

0.00-10.00	10.00-20.00	20.00-30.00	30.00-40.00	40.00-50.00
50.00-60.00	60.00-70.00	70.00-80.00	80.00-90.00	90.00-100.00

Figure 6.5 (above) shows the following: As was the case for the results from networks trained using Symmetric Local Learning, the plot pairings a-b and c-d are for patterns of bias 0.5 and 0.9 respectively. The left-hand plots in each pairing represent networks in which random removal was performed while the right-hand plots represent those in which smallest-first removal was used.

The results are very similar to those for Symmetric Local Learning. We again see the change in pattern bias from 0.5 to 0.9 making little difference to the way

in which the level of pattern stability changes with respect to the increasing loading and the decreasing connectivity.

As was the case for Symmetric Local Learning, it is again evident that it is not possible to remove as many connections at high loadings as at low loadings whilst maintaining a high level of pattern stability.

Regardless of the pattern bias the advantage of using smallest-first dilution is again apparent with the Blatt and Vergini learning rule. Comparing the plots in the pairings a-b and c-d it can be seen that the increase in the area of the purple region representing maximum stability is significant when moving from random to smallest-first removal.

A key difference between the plots of pattern stability for Symmetric Local Learning and Blatt and Vergini is rate at which pattern stability declines at high loadings. At a loading of α=0.95 (95 patterns), Symmetric Local Learning-trained networks exhibited approximately 90-100% stability at up to 30% removal for unbiased patterns and up to 35% removal for biased patterns. At the same loading, Blatt and Vergini-trained networks have 90-100% stability at only up to 5% removal. It is concluded therefore, that networks trained using Symmetric Local Learning have greater resilience to smallest-first removal at high loading than those trained using the Blatt and Vergini algorithm.

6.4.2.2. *Attractor Performance*

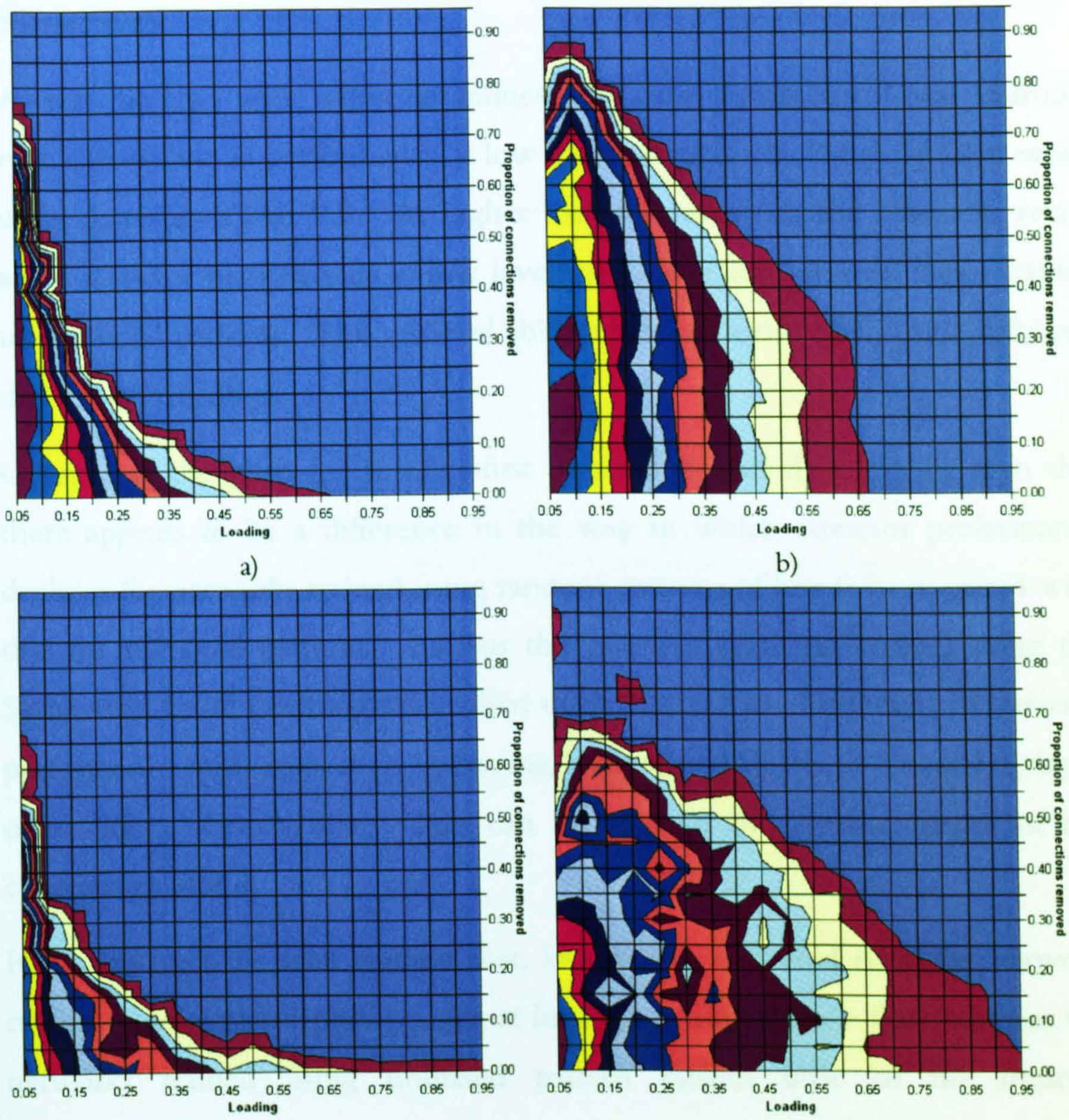


Figure 6.6: The manner in which attractor performance changes with respect to increasing network load and decreasing levels of connectivity. The individual plots represent a) pattern bias 0.5, random removal; b) pattern bias 0.5, smallest-first removal; c) pattern bias 0.9, random removal; d) pattern bias 0.9, smallest-first removal.

Figure 6.6 (above) shows the following: The plot pairings a-b and c-d are as for pattern stability with the left-hand plots in each pair representing networks in which the random removal strategy was used while the right-hand plots represent networks where connections were removed using the smallest-first strategy.

Comparing the plots for random removal (a and c) it can be seen that the change in pattern bias from 0.5 to 0.9 has a small effect on the way in which

attractor performance declines with increasing loading and decreasing connectivity.

As was the case with networks trained using the Symmetric Local Learning rule, the decline in performance at low loadings takes place slightly faster when using patterns of bias 0.9. At higher loadings however, the networks retain some attractor performance at low levels of connection removal when trained using biased patterns. This retrieval ability does not exist when using unbiased data at such loadings.

Comparing the plots for smallest-first removal (b and d) it can be seen that there appears to be a difference in the way in which attractor performance declines for networks trained using random patterns of bias 0.9 compared with that for unbiased patterns. As was the case for networks trained using the Symmetric Local Learning rule, when using biased data the decline in attractor performance with respect to increasing loading and decreasing connectivity is often not a smooth one. This can be seen in the chaotic nature of the contours in plot d.

It can be again be seen though that, in the case of biased data, the network retains some attractor performance at loadings greater than 0.67 whereas the networks trained using unbiased pattern possess little to no attractor performance beyond that point. So, as was evident with networks trained using Symmetric Local Learning, it again appears that networks learning biased patterns are more resilient to a decrease in connectivity than those learning unbiased patterns when training using the Blatt and Vergini rule.

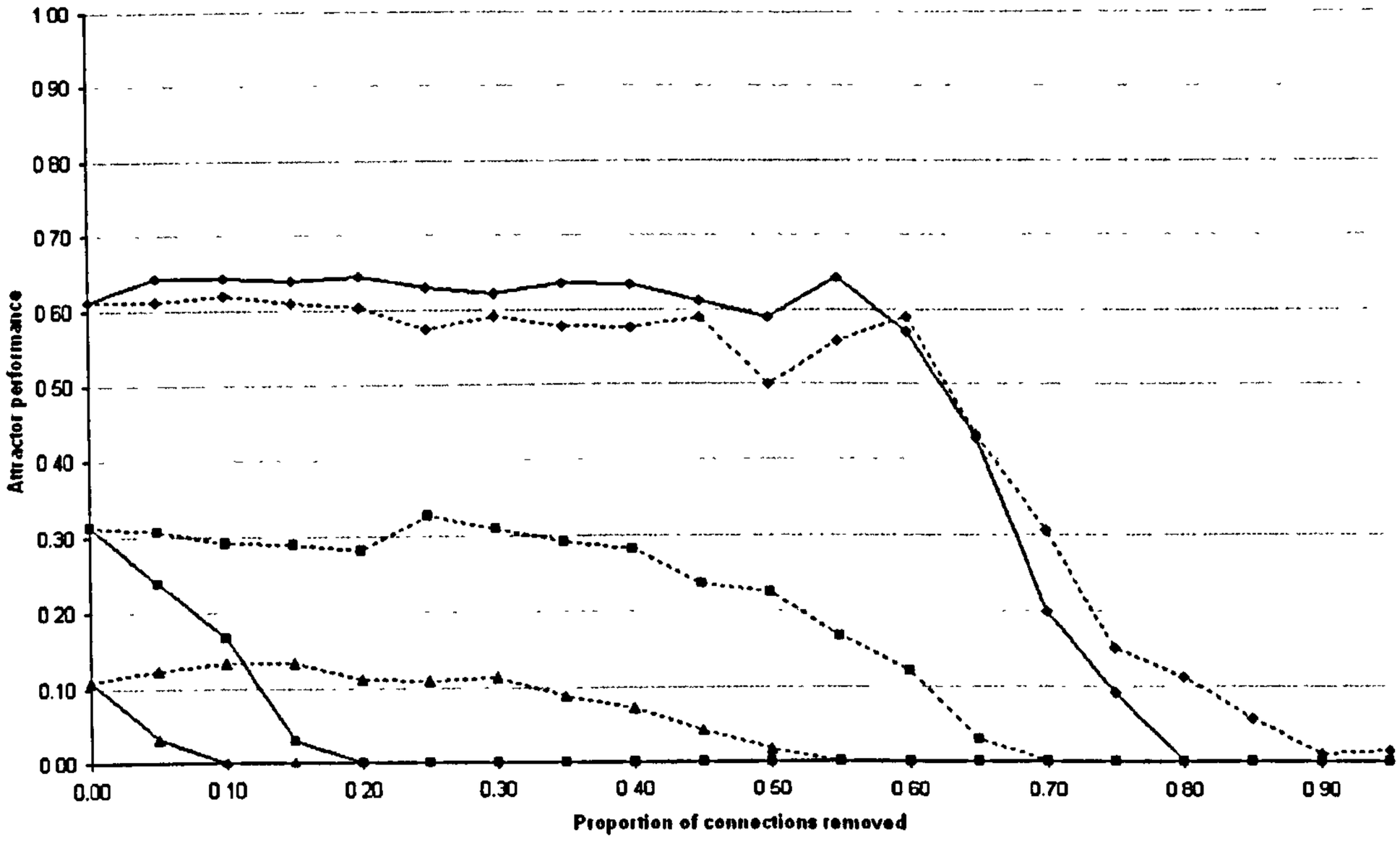


Figure 6.7: The decline in attractor performance (R) for a number of fixed loading points (0.05, 0.30, and 0.50) using patterns of bias 0.5. The results of both random removal and smallest-first removal are superimposed for comparison.

Key: Definitions of line styles representing levels of pattern load and type of removal strategy

5 patterns, random removal	5 patterns, smallest-first removal
30 patterns, random removal	30 patterns, smallest-first removal
50 patterns, random removal	50 patterns, smallest-first removal

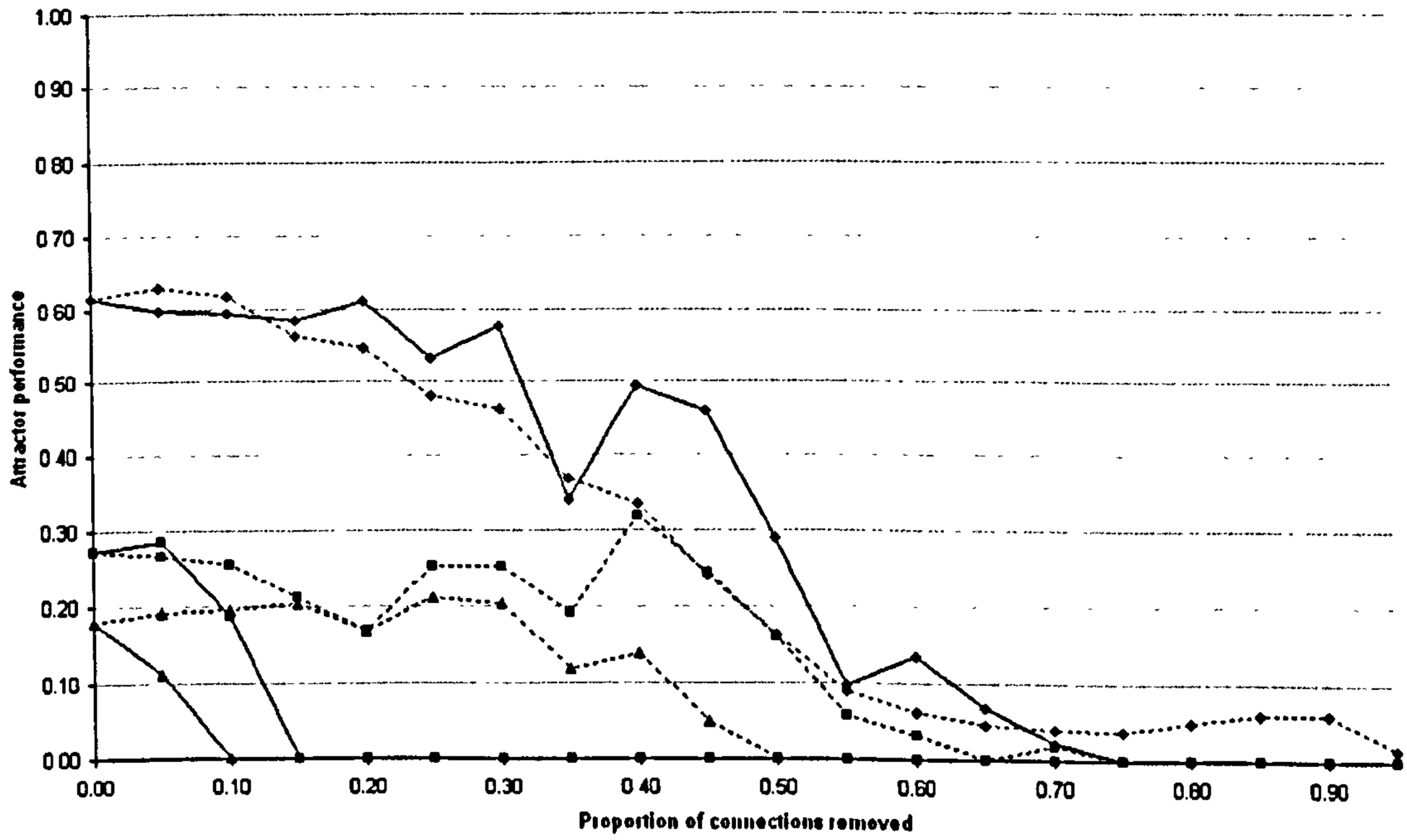


Figure 6.8: The decline in attractor performance (R) for a number of fixed loading points (0.05, 0.30, and 0.50) using patterns of bias 0.9. The results of both random removal and smallest-first removal are superimposed for comparison.

Key: Definitions of line styles representing levels of pattern load and type of removal strategy

5 patterns, random removal	5 patterns, smallest-first removal
30 patterns, random removal	30 patterns, smallest-first removal
50 patterns, random removal	50 patterns, smallest-first removal

Figures 6.7 and 6.8 illustrate the way in which attractor performance falls as the proportion of connections being removed increases for networks trained using the Blatt and Vergini learning rule with patterns of bias 0.5 and 0.9 respectively. Each graph shows results for both random and smallest-first removal (represented by solid and dashed lines respectively) at loadings of α=0.05, 0.30, and 0.50.

At a loading of 0.05, both graphs show that the attractor performance is often better when removing connections from the network at random. This appears to be the case for up to approximately 65-70% removal of connections. Beyond this point, smallest-first removal becomes more advantageous.

The situation changes significantly at higher pattern loads. For unbiased patterns at a loading of 0.30 smallest-first removal allows around 65-70% of connections to be removed before attractor performance disappears completely compared with 15-20% removal when performed randomly. At a loading of 0.50 this falls slightly to 50-55% for smallest-first versus 10% for random removal.

As one might expect from the earlier evidence that the level of bias makes little difference to the performance, the values for biased patterns at loadings of 0.30 and 0.50 are very similar to those for unbiased. The increased irregularity in the contours of figure 6.6(d) does not manifest itself in as chaotic a decline in performance as might be expected.

6.5. Discussion

When read along a line of constant pattern load, the spacing between the contours of the plots showing pattern stability against increasing load and declining connectivity (figures 6.1 & 6.3) indicates the rate of decline of the stability of the patterns learnt by the network. A wider band of colour indicates a slower reduction in stability. All the plots showed that when stability began to be lost, the networks experienced a rapid fall to a point where stability was 0-10% of the trained patterns rather than a gradual decline in stability. This is indicated by the plots having a large band of colour representing 90-100% stability before a point is reached at which a number of narrow bands appear in succession representing a rapid fall in stability before another wide band representing 0-10% stability appears.

Contrasting the removal strategies shows that random removal is clearly out-performed by the smallest-first method. This is shown by the area of the plots coloured purple and representing 90-100% stability in figures 6.1 and 6.3.

Comparing between the training algorithms reveals another result. When employing smallest-value-first removal the Symmetric Local Learning algorithm consistently outperforms the Blatt and Vergini rule with regard to the point at which networks begin to lose stability at higher loading/removal levels. It can be seen that at a loading of $0.95N$ the networks trained using the Blatt and Vergini rule have lost more than 10% of their patterns with a loss of only 5-10% of their connections. The networks trained using Symmetric Local Learning do not experience this until the level of connection removal reaches 30-40%. This holds true regardless of level of bias in the training patterns.

Examining the plots of attractor performance against loading and connectivity (figures 6.2 & 6.4) two important observations can be made. The first of these is that looking from left to right across the plots, i.e. from random to smallest-value-first removal strategies, it can be seen that improvement in performance can be gained by targeting connection removal towards those with values closest to zero rather than selecting them at random. The second point of note is that altering the level of bias in the patterns being stored seems to make relatively little difference to the attractor performance of the network as can be seen by comparing the top and bottom plots of each set.

This second point is perhaps unsurprising in the case of the Symmetric Local Learning algorithm given that the maximum loading tested is well below the notional maximum capacity of $2N$ patterns (c.f. chapter 2) but more so in the case of the Blatt and Vergini rule where the maximum capacity is N-1. The conclusion that networks trained using Symmetric Local Learning are more resilient is reinforced by the increased robustness against connection removal at high loadings when considering pattern stability.

6.6. Conclusions

The results presented show that, when trained used high-performance learning rules, Hopfield-type networks can be highly resilient in terms of maintaining high levels of performance even after considerable levels of damage has been done to the connectivity of the network. This resilience is, however, highly dependant on both the way in which a network's weight matrix has been generated and the manner in which the connections have been removed. The results show that of those tested, the most effective learning rule is Symmetric Local Learning and the best synapse removal strategy is that of smallest-value-first.

The results for random removal concur with that of Sompolinsky (1986). Somplinksy showed that, for networks trained using the Hebb rule, pattern stability fell linearly with the proportion of connections removed from the network. For the networks used in this investigation this is also the case. The level of removal at which stability begins to be lost is dependant on the actual pattern load on the network but once instability begins its rise is approximately linear with increasing connectivity removal. For non-trivial levels of loading this linear decline is also true for attractor performance.

To conclude, it was demonstrated that a significant amount of connectivity can be removed from a trained network without adversely affecting either the pattern stability or the attractor performance to any great degree. The reduced connectivity brings benefits in terms of the storage requirements of such networks when implemented in software and has implications for both hardware implementations and the biological plausibility of Hopfield-type associative memories.

7. DEVELOPMENT AND ANALYSIS OF NON-RANDOM TRAINING DATA

7.1. Introduction

This chapter is concerned with the development of non-random training data and the analysis of the characteristics of that data. The purpose of generating this type of data is to attempt to simulate what Müller et al. (1993) term the unfavourable correlations inherent in natural or real world data sets that arise from the spatial and temporal continuity of nature. According to Müller et al., it is these correlations that cause standard fully-connected models to fail prematurely.

In order to test the hypothesis laid out in chapter 5 regarding the effect of structured or correlated data on the ability of an associative memory to efficiently store and effectively recall that data it is important that the nature of the data being employed be well understood.

The requirements of a training set comprised of this data are the same as those suggested by Müller et al. in that it should possess:

a) High inter-pattern correlation: The patterns should be relatively similar to each other. In practice, this means that identically positioned bits in a set of patterns will often have the same value throughout the pattern set.

b) High site correlations within each pattern: Within each of the patterns, there should be areas where the majority of bits are the same value. This would be represented by blocks of the same colour within individual patterns.

Two sets of training patterns were created. All the generated patterns were 400 bits in length. The new patterns are four times bigger than those used previously in this work and the larger dimensions will permit greater flexibility in establishing structured connectivity topologies in networks of the same size in later investigations. To facilitate the development of such topologies the training patterns are 2-dimensional representations of the 400-dimensional inputs. Enforcing spatial proximity within the training patterns gives greater meaning to a non-fully-connected system of connectivity through the arising of the potential for reflecting training pattern structure in the connectivity topology. This is in contrast to the fundamentally unstructured nature of the traditional Hopfield (1982) network.

The first set of training data takes the form of solid geometric shapes placed at random within the 2-dimensional representation of a training pattern. The second data set comprises images taken from computer character sets, or fonts.

7.2. Generating Non-Random Data

This section presents a description of the way in which the different types of non-random data were created. Initially, an attempt was made to source suitable 2-colour images from clip-art resources. Such images exist but possess two distinct disadvantages. Firstly, the images tend to be quite large by comparison with the size of the networks being used in this work. Resizing the images caused a lot of image detail to be lost and so the natural correlations present were likely to be have been destroyed in the process. The second disadvantage arose from the fact that where the images were of a suitable size, they were often not square. Scaling the images to be square distorted the images in such a way that the correlations were no longer the same as those that occurred naturally.

7.2.1. Geometric Data

The geometric data set is generated by placing, at random locations, a number of solid geometric shapes within the 20 by 20 pixel training pattern. The shapes used are triangles, squares, and circles. The choice of shape to place each time is also random. The shapes are permitted to overlap and are clipped if a shape would overrun an edge.

Some example patterns are show below:

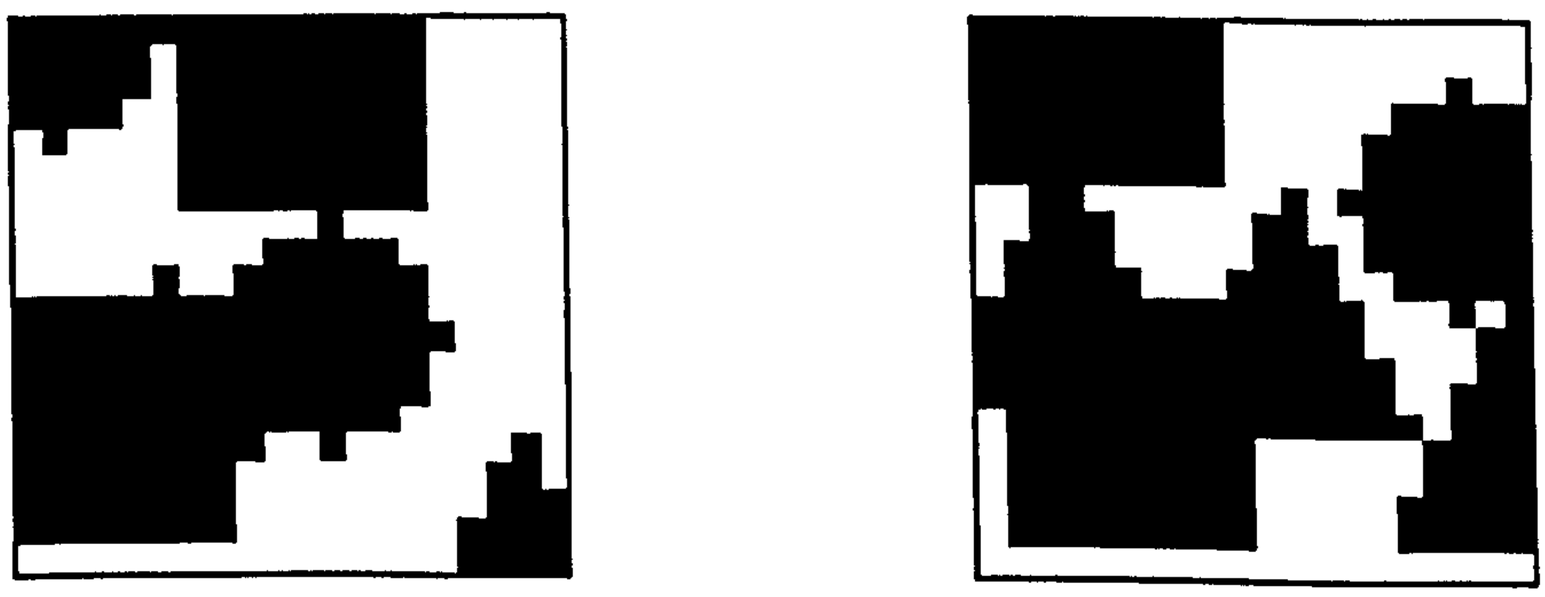


Figure 7.1: Two examples of training patterns based on the generated geometric data.

The patterns shown in figure 7.1 have large areas that are the same colour. This provides us with one of our requirements of the data in that the patterns have high site correlations. If a bit is picked at random from one of the above patterns, there is a high likelihood that its neighbours will be the same colour.

The other requirement (the inter-pattern correlation) is more complex to analyse and its fulfilment or otherwise is examined in further detail in the section on analysis of pattern characteristics, below.

A selection of the geometric training data is presented in appendix B.

7.2.2. Character Data

The character data set is generated by scaling images of letters from computer character sets into the 2-dimensional training pattern representation. Although the problems with scaling the data were described earlier in the context of monochrome clip-art images, scaling the relatively simple characters that make up this training data causes no such difficulties.

Examples of patterns generated this way are shown below:

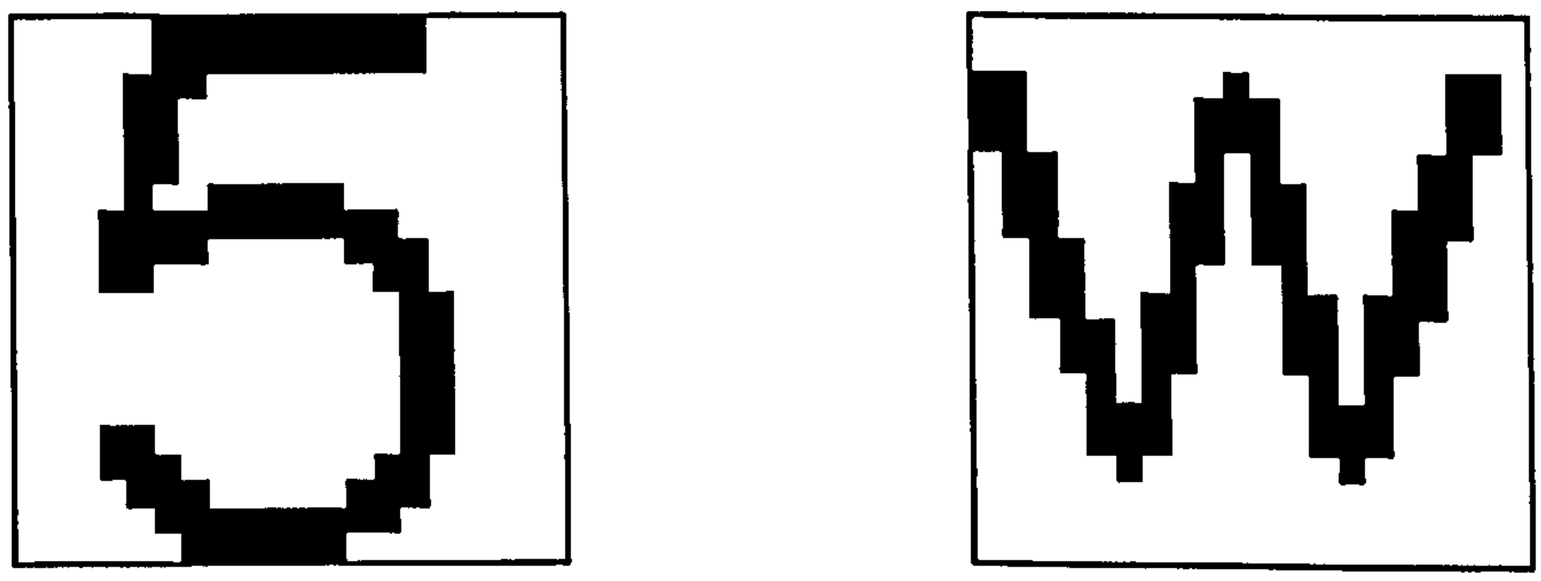


Figure 7.2: Two examples of training patterns based on the character data.

As with the geometric data seen previously it is apparent that the patterns shown in figure 7.2 possess large areas of the same colour which again fulfils the requirement for high site correlation.

The full set of character-based training data is presented in appendix C.

7.3. Analysis of Training Pattern Characteristics

The importance of understanding the underlying nature of the training data has been mentioned previously. This section provides information about four methods by which information about the data may be obtained. These techniques are: measuring the bias of the training patterns, calculating local correlation across the training set, calculating site correlation within the patterns, and measuring site activity across the set. All four techniques are described in detail below.

7.3.1. Measuring the Bias of a Training Set

The bias of a set of training patterns gives a measurement of how much the bits that make up that set favour a particular value. In the case of the bipolar patterns employed in this work, that value may be +1 or -1.

Unbiased patterns, those whose bits may take the value +1 or -1 with equal likelihood, have a bias of 0.5. The bias reflects the probability that any bit, chosen at random from patterns in the training set, will have the value +1.

To illustrate this, some example patterns are shown below:

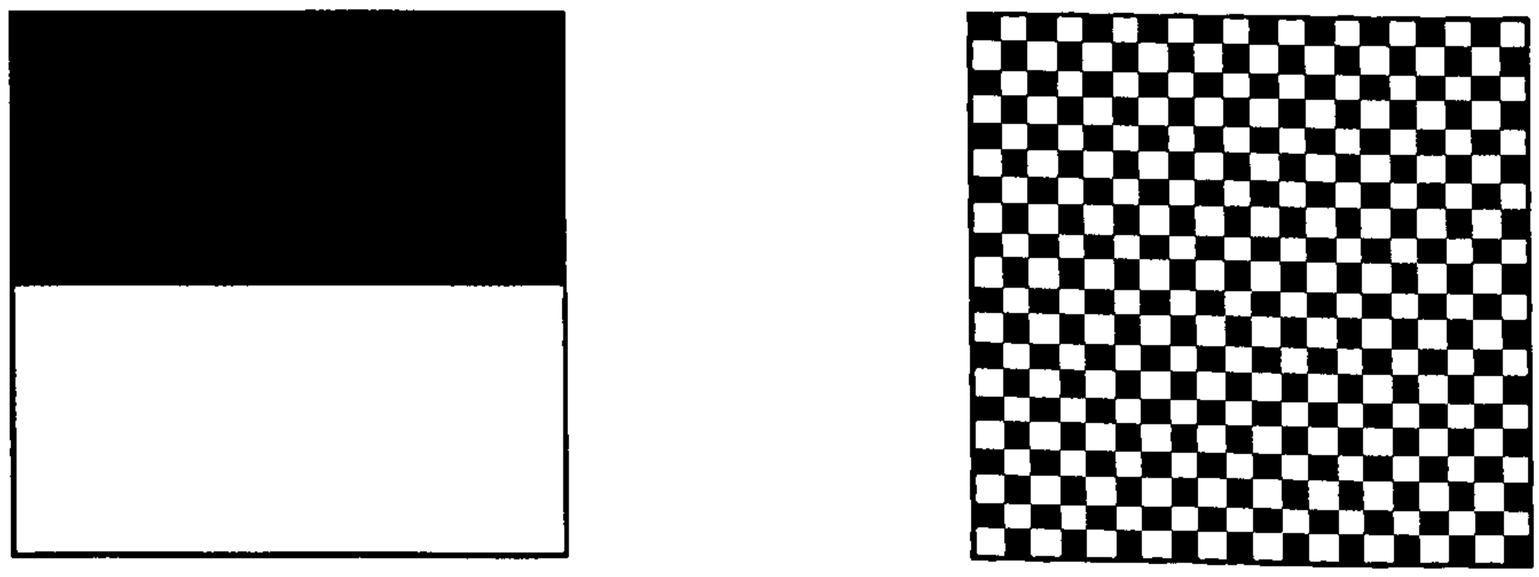


Figure 7.3: Two example patterns with bias 0.5.

The example patterns shown in figure 7.3 (above) are both unbiased. The convention throughout this work has been to portray +1 bits as black, and -1 as white.

It should be recognised that the bias values are symmetric about the value 0.5. That is to say, a pattern with a bias value of 0.2 can be considered to be as heavily biased as a pattern with bias 0.8. One pattern will be heavily biased to bit values of +1 and the other biased to bit values of -1.

The bias of a set of training patterns gives an approximate indication, especially in the case of random data, of the complexity of the dataset.

7.3.2. Calculating the Local Correlation within a Training Pattern

The level of global correlation of a training pattern indicates how similar, on average, each bit is with the other bits in the pattern. In contrast, the local correlation of a training pattern provides a measure of how similar each bit is to those in its immediate vicinity. Both calculations are averaged over all the bits in the pattern.

In order to calculate local correlation we must have some definition of a locale for which it should be calculated. For the purposes of this calculation, the locale is defined as being a square neighbourhood around some specified bit.

For example:

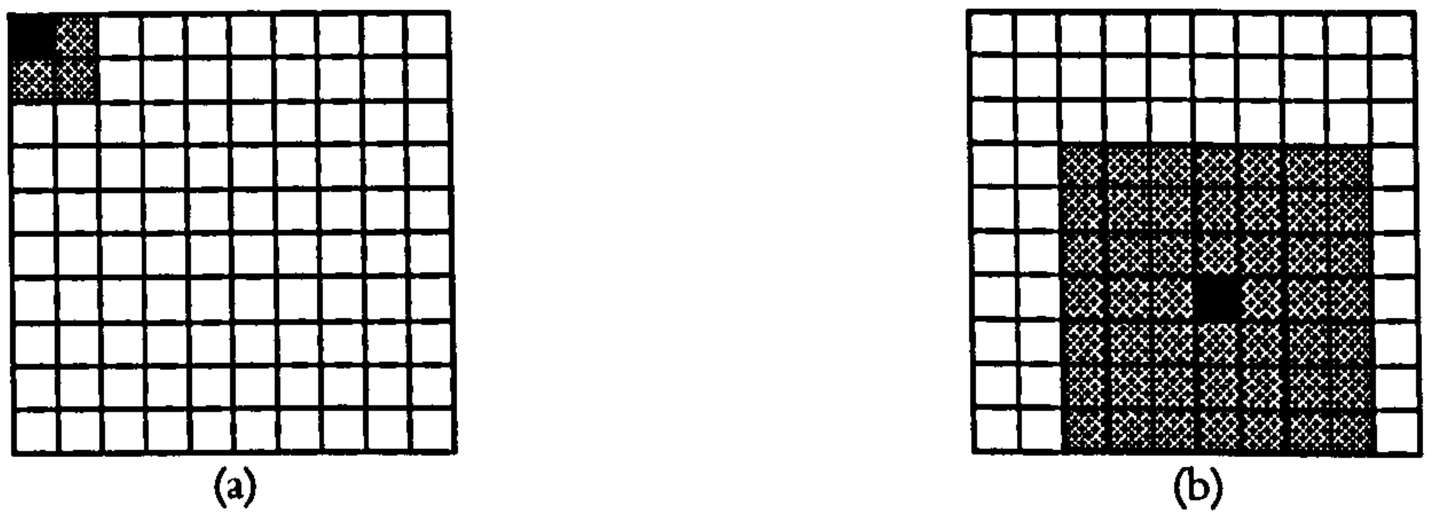


Figure 7.4: (a) An example of a bit with a neighbourhood size (d) equal to 1. (b) An example of a bit with a neighbourhood size (d) equal to 3.

In figure 7.4(a) a neighbourhood around a corner bit is shown. The size of the neighbourhood is defined by the distance of the furthest non-diagonal bit and in this case the distance, *d*, is 1. Figure 7.4(b) shows a neighbourhood around a more central bit; this time the distance is greater (d=3). The slightly greater Euclidian or city-block distances of bits set at a diagonal is ignored for the purposes of simplicity of definition.

Consider the simplest case of a single pattern in which all bits have the same value. We would expect this measure to indicate maximal correlation. The level of correlation is denoted by a value between 0 and 1. A value of 0 will mean there is no local correlation present in the data while a value of 1 will mean the opposite, that the correlation is as high as it can be.

The correlation for a single bit, i, is calculated as follows:

$$C_i = \frac{1}{\#Z_i} \sum_{j \in Z_i} \Theta(\xi_i \xi_j) \tag{7.1}$$

Where Z_i is the set of indices of the bits comprising the neighbourhood of unit i and Θ is the unit Heaviside function. C_i is effectively the proportion of a bit's neighbours possessing the same value as that bit. If all the neighbouring bits have the same value as the bit for which the correlation is being measured then C_i will equal 1. Correspondingly, if all the neighbouring bits are a different value then C_i will equal 0 though, in practice, the corner and edge bits ensure this will only occur as $N \rightarrow \infty$.

Having measured the local correlation of one bit, it remains to calculate the local correlation for all others in the pattern. The mean of these values is taken to be the overall level of local correlation present in the pattern.

The level of global correlation is calculated in exactly the same way except all bits are considered to be in the neighbourhood of the one for which the measure is currently being calculated. The mean is again taken and this value is the level of global correlation.

As the level of bias of a pattern increases, so should the level of correlation. For random patterns, the level of local and global correlation should be very similar as the active sites in each pattern will be evenly distributed. For patterns in which the data is more structured, such as the examples of geometric and character data shown in §7.2, it would be expected that the local correlation level would be significantly greater than that of global correlation.

A high level of local correlation is important because, as described in chapter 5, it implies that if the network is constructed with a connectivity topology resembling the neighbourhood locales then a high degree of correlation between the desired output for a neuron, and its inputs, will arise.

The fact that a pattern *is* locally correlated is often intuitively evident from simply seeing the pattern. More important however, is that a set of patterns are highly correlated for the same locales in each pattern as this will give rise to an advantageous environment as described by Lopez et al. (1995). The next measure described identifies whether or not cross-pattern local correlation exists.

7.3.3. Calculating the Level of Local Correlation across a Training Set

It was illustrated in chapter 5 that, for perceptrons, the more similar the patterns within a training set are to each other, as long as they share the same output value, the easier it becomes to learn them. It is possible to measure this similarity by calculating the level of local correlation across a training set. For local connectivity to be advantageous, the level of local correlation must be greater than that of global correlation.

The definition of locality is the same for this calculation as that which was used for single patterns. The aim of the local version of this measure is to calculate the mean local correlation of pattern subsets where the subsets are determined by a central bit and those comprising a square neighbourhood around it. A subset of a pattern's bits, defined in this way, has been termed a *sub-pattern*.

To calculate the correlation we first produce a Hebb-style matrix representing the mutual pattern correlations. An N-by-N matrix is defined and termed $\mathbf{T}$. The matrix element T_{ij} represents the proportion of patterns in which bits i and j have the same value. The elements are calculated as follows:

$$T_{ij} = \frac{1}{P}\sum_{\mu=1}^{P}\Theta\left(\xi_i^{\mu}\xi_j^{\mu}\right) \tag{7.2}$$

Once the correlation matrix has been created the global correlation may be calculated as follows:

$$Correlation_{global} = \frac{1}{N^2 - N}\sum_{i,j=1,i\neq j}^{N} T_{ij} \tag{7.3}$$

The resulting value should be almost identical to the overall level of bias present but as measuring the bias includes self-correlation some small difference between the values will exist.

The local correlation is calculated by defining a square neighbourhood around a specified bit and proceeds as for global correlation but restricted to the neighbourhood.

$$Correlation_{local_i} = \frac{1}{\#Z_i}\sum_{j\in Z_i} T_{ij} \tag{7.4}$$

As with the measure for a single pattern, Z_i is the set of indices of the bits comprising the neighbourhood of unit *i*. T_{ij} is the correlation matrix calculated according to equation (7.2).

7.3.4. Measuring Site Activity across a Training Set

The site activity across a set of training patterns gives a simple indication of the importance of a particular bit in terms of its overall contribution in terms of information. This measure was used by Stiefvater et al. (1993) in order to determine redundant synapses which may be safely cut. A value b_i is defined as:

$$b_i = \frac{1}{P}\sum_{\mu=1}^{P}\Theta\xi_i^{\mu} \tag{7.5}$$

where P is the number of patterns in the training set and Θ is again the unit Heaviside function. The value b_i can be thought of as the bitwise bias of the training set.

A low value for b_i means that, on average, the majority of the patterns in the training set have a -1 value at position *i*. Conversely, a high b_i means most of the patterns have a +1 value at position *i*. Stiefvater et al. note that weights leading to neurons with a value of b_i close to the average bias, b, for the entire training set are the most important ones and should be kept. These weights are important as they are providing the information required to allow those neurons with the most difficult classification tasks to perform accurately.

The bitwise bias also indicates the degree of cross-pattern similarity at a particular site. If an individual bit has a bias of 0.9 across the set of training patterns then it is known that in 90% of the patterns that bit has a value of +1. This would represent a very high degree of similarity between the patterns at that site.

7.4. Results of Training Pattern Analysis

This section presents the results of analyses performed upon the training data. The analyses are: training set bias, cross-pattern local correlation, and site statistics or bitwise bias. Results are not presented for the measurement of local correlation on a per-pattern basis as the presence or otherwise of this characteristic is adequately given by the cross-pattern local correlation measure.

7.4.1. Training Set Bias

As mentioned previously (c.f. §7.3.1), the bias of a training set gives a rudimentary indication of its structure. The bias of the geometric and character data was measured over 5 sample sets of 50 patterns each and the mean bias was calculated.

Data type	Bias
Geometric data	0.52
Character data	0.20

Table 7.1: Training data set bias for geometric and character data.

The figures given in table 7.1 indicate the level of bias in each of the constructed data sets. Geometric data has a bias of 0.52 indicating that, on average, about half the bits in each pattern have a value of 1. Character data, with a bias of 0.2, has one fifth of its bits having a -1 value and four fifths being equal to 1.

7.4.2. Cross-Pattern Local Correlation

As noted earlier in chapter 5, although Gardner (1988) presented evidence of the fact that the capacity of perceptrons rises with pattern bias, Lopez et al. (1995) demonstrated the added importance of the similarity between the training patterns. If it can be demonstrated that correlation is greater at a local level for natural data than at a global level then connectivity matching the locale at which correlation is greatest may provide capacity and performance benefits. An example demonstrating why this should be the case is presented later in chapter 8.

7.4.2.1. Geometric Data

The following sequence of histograms shows the frequency with which various correlation values occur for a set of geometric data at neighbourhood distances, *d*, of 1 to 5. The value for the global correlation in each case is 0.5161.

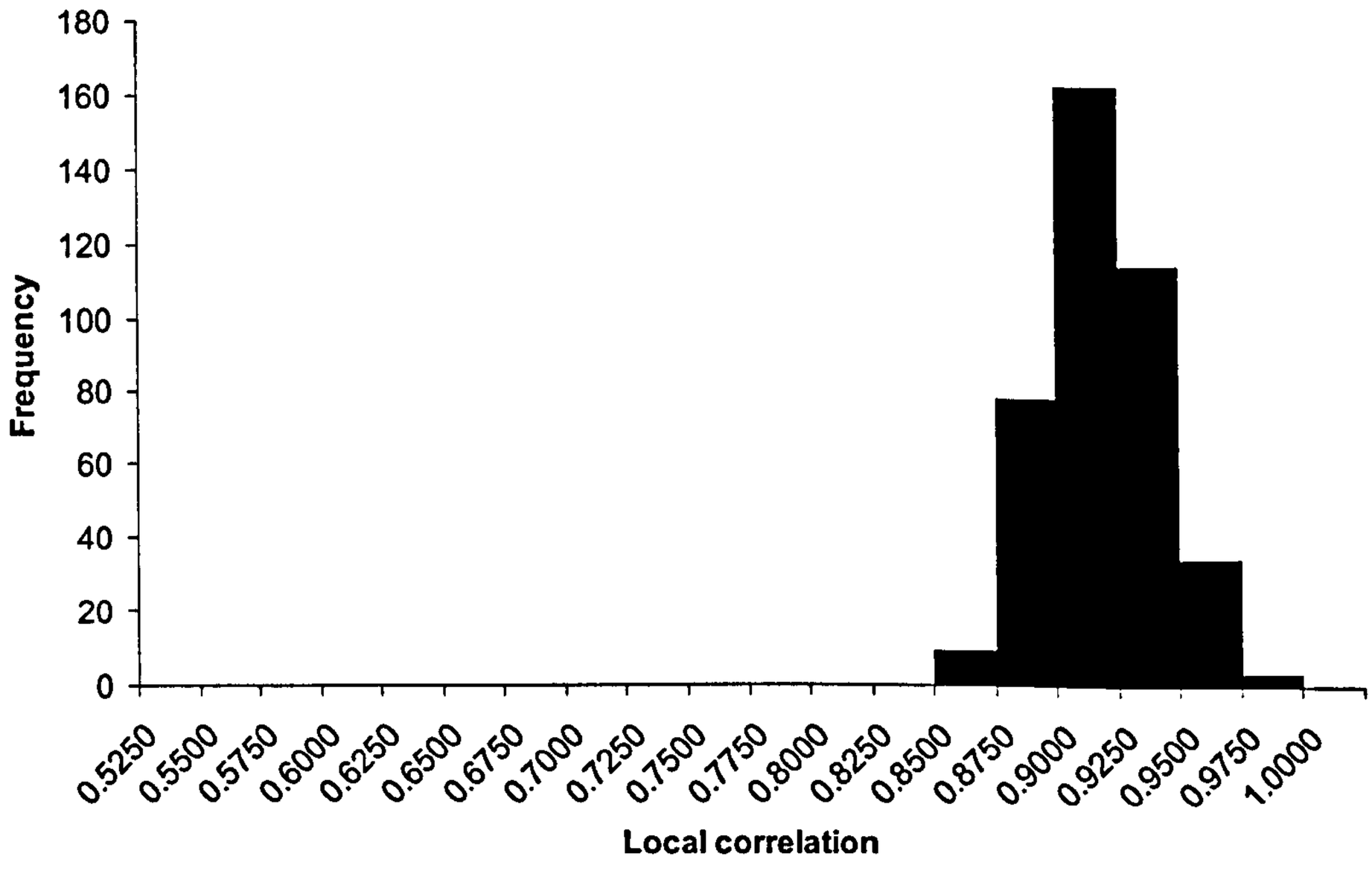


Figure 7.5: Frequency distribution of the cross-pattern local correlation values for geometric data at a neighbourhood distance, d=1.

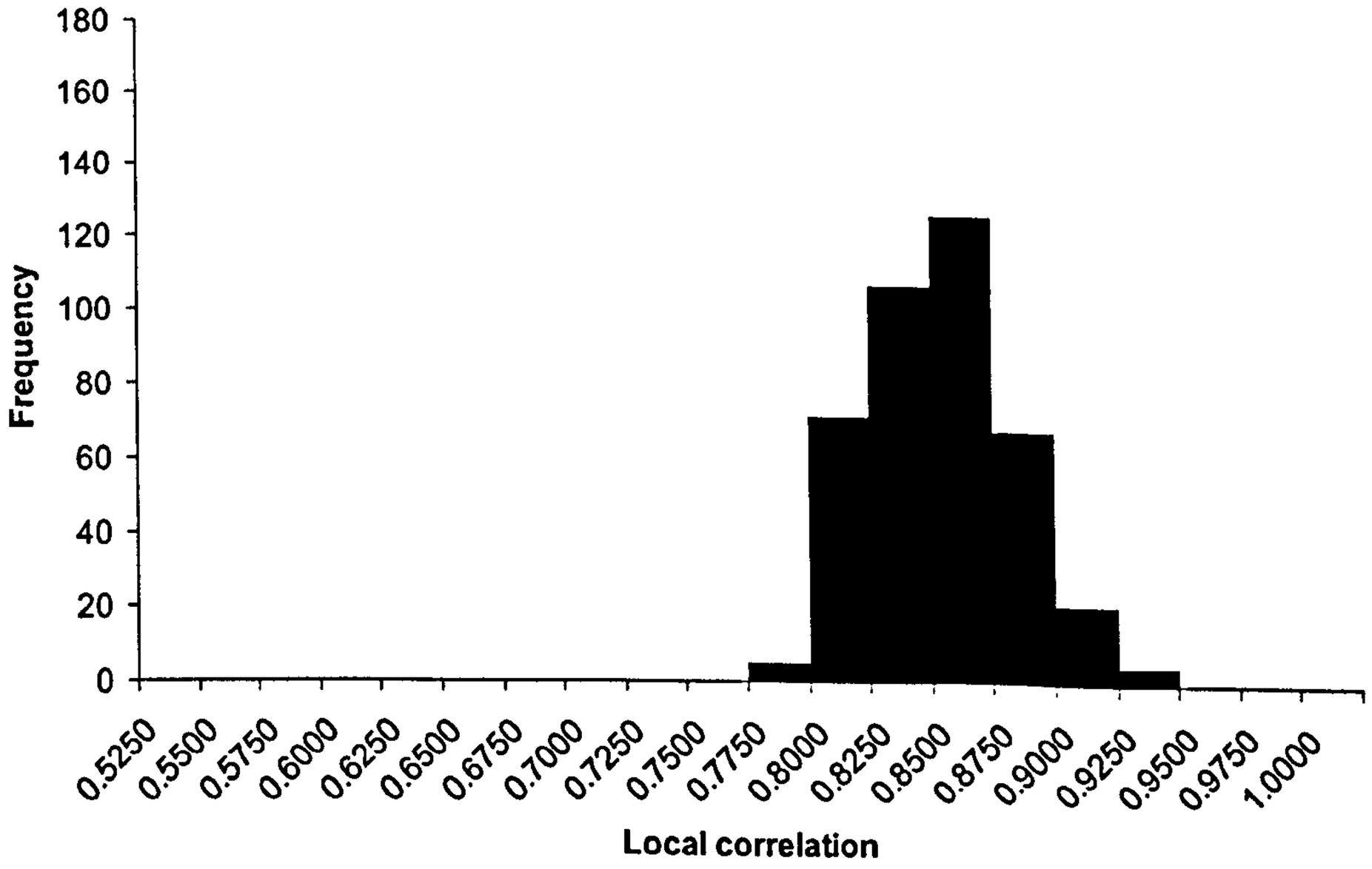


Figure 7.6: Frequency distribution of the cross-pattern local correlation values for geometric data at a neighbourhood distance, d=2.

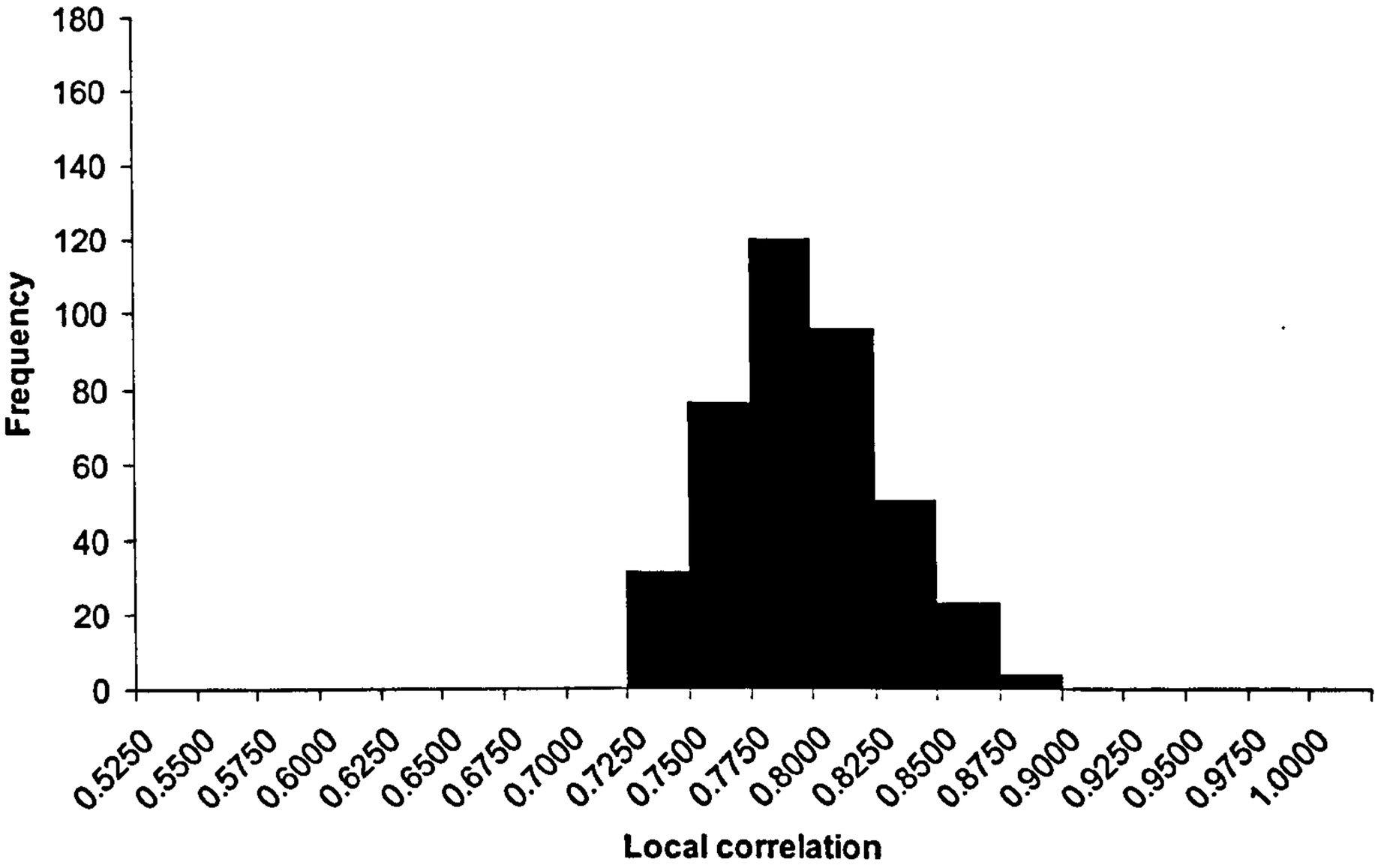


Figure 7.7: Frequency distribution of the cross-pattern local correlation values for geometric data at a neighbourhood distance, d=3.

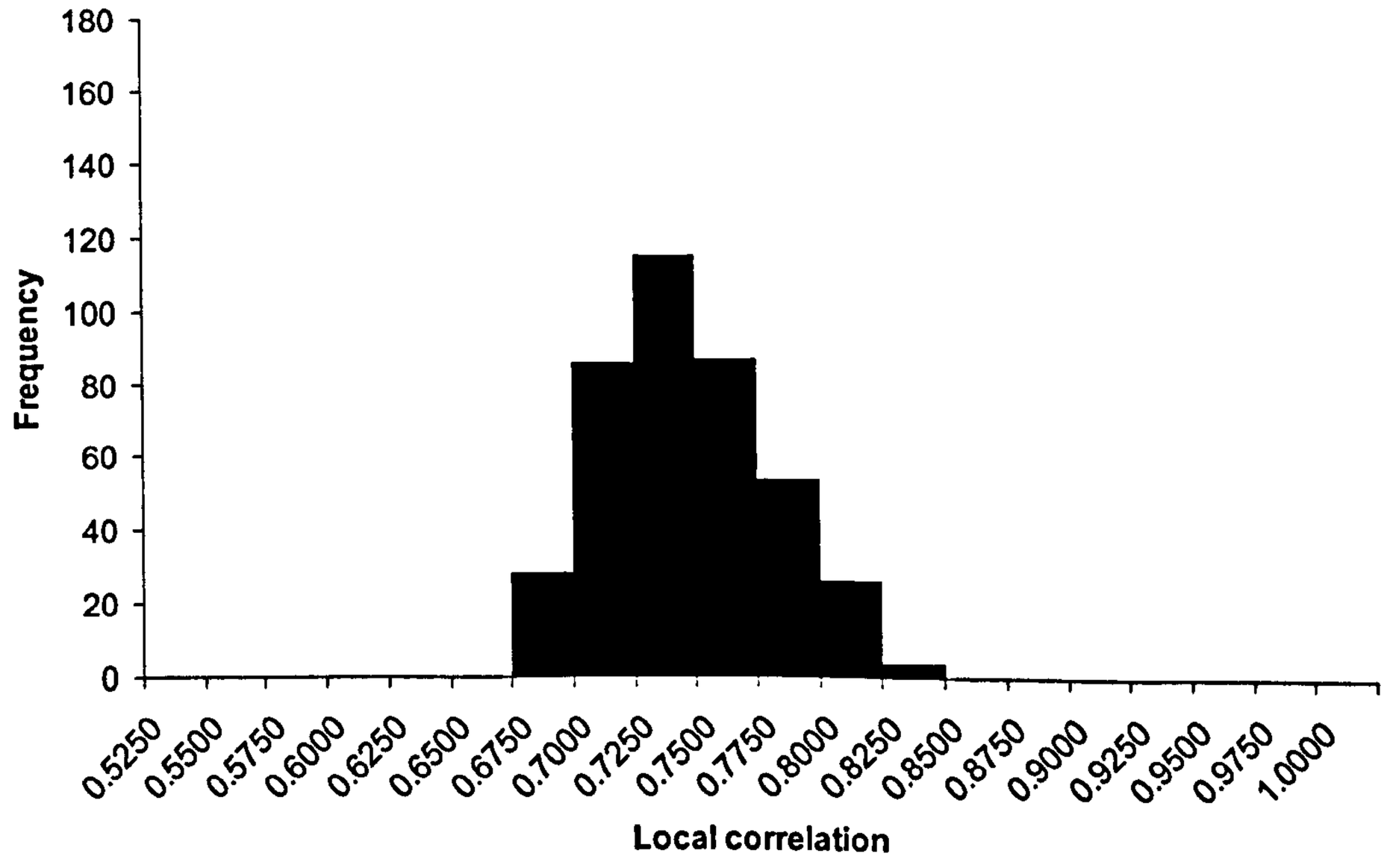


Figure 7.8: Frequency distribution of the cross-pattern local correlation values for geometric data at a neighbourhood distance, d=4.

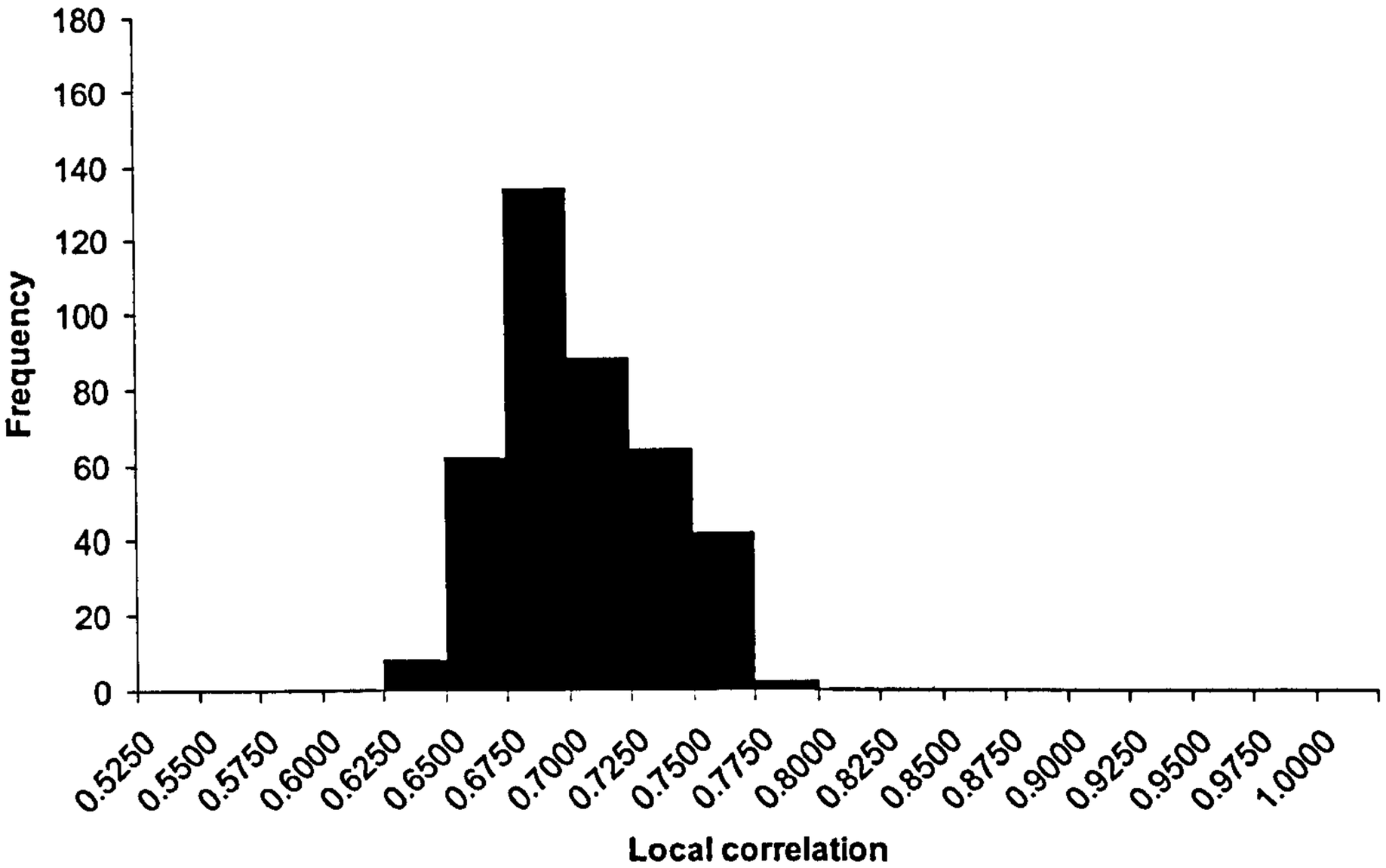


Figure 7.9: Frequency distribution of the cross-pattern local correlation values for geometric data at a neighbourhood distance, d=5.

The sequence of histograms above show that, as the sub-pattern area over which the correlation is measured increases, the values for local correlation move towards the measured value of global correlation.

Furthermore, it is apparent that the greatest degree of local correlation exists when measured using sub-patterns forming a neighbourhood of distance d=1. It must be noted that even at the maximum sub-pattern size tested, those forming a neighbourhood of d=5, the level of local correlation at every bit is still above that of global correlation. This occurs because the correlation measure at d=5 incorporate the values for local correlation at previous neighbourhoods.

The way in which the level of local correlation falls with respect to the increasing neighbourhood size is best illustrated by plotting the mean local correlation values shown in table 7.2.

Neighbourhood size	Mean local correlation
1	0.89 (s.d.=0.02)
2	0.83 (s.d.=0.03)
3	0.77 (s.d.=0.03)
4	0.72 (s.d.=0.03)
5	0.68 (s.d.=0.03)

Table 7.2: Mean local correlation values at various neighbourhood sizes for geometric training data.

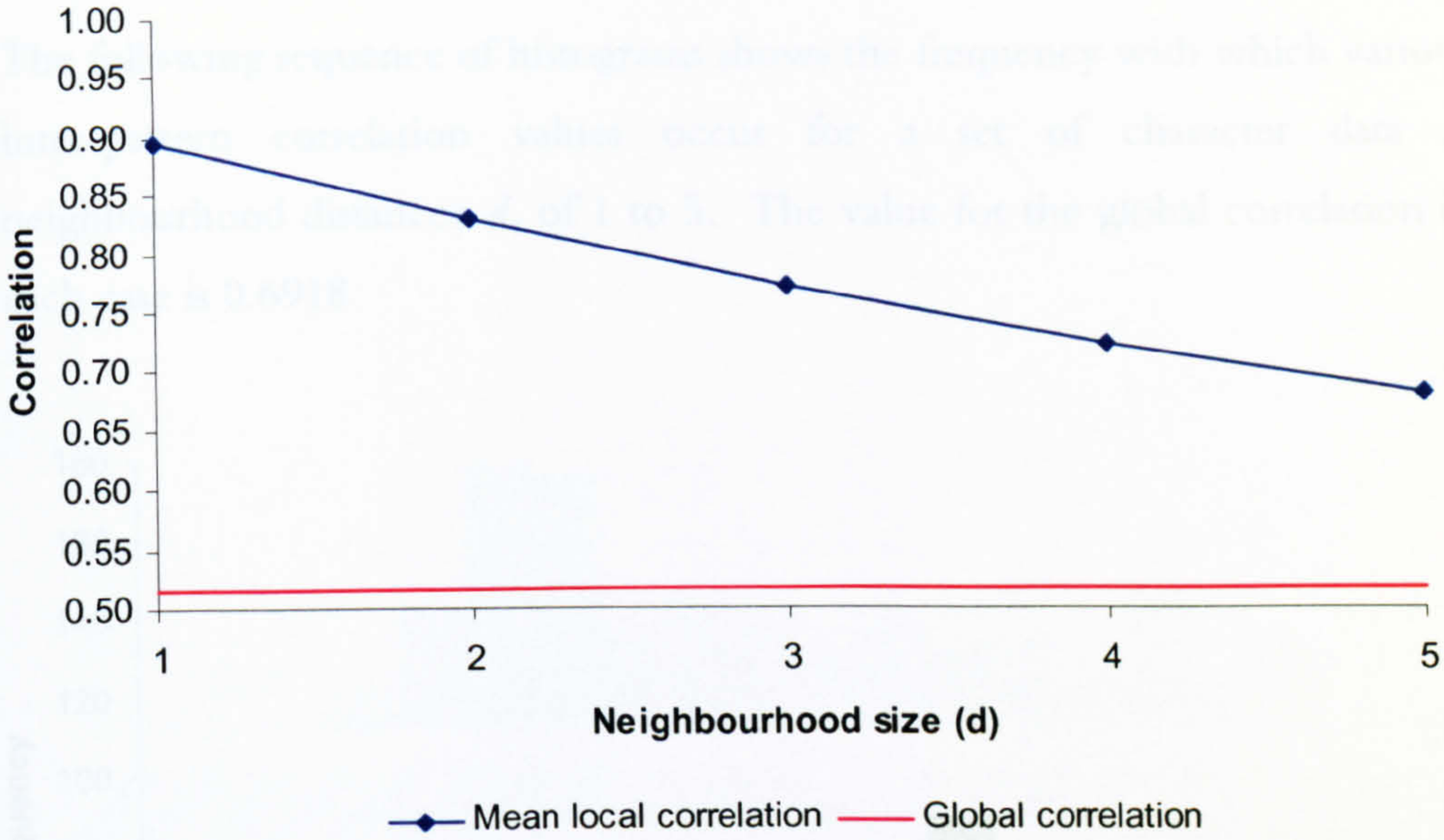


Figure 7.10: Mean local correlation against sub-pattern neighbourhood size for geometric training data. The level of global correlation is shown for comparison.

Figure 7.10 (above) shows that the level of global correlation, indicated by the red line, lies at 0.5161. The level of local correlation appears to fall linearly with respect to the increasing neighbourhood size from a maximum value of 0.89 at a neighbourhood size of 1 to 0.67 at a neighbourhood size of 5.

7.4.2.2. Character Data

The following sequence of histograms shows the frequency with which various inter-pattern correlation values occur for a set of character data at neighbourhood distances, d, of 1 to 5. The value for the global correlation in each case is 0.6918.

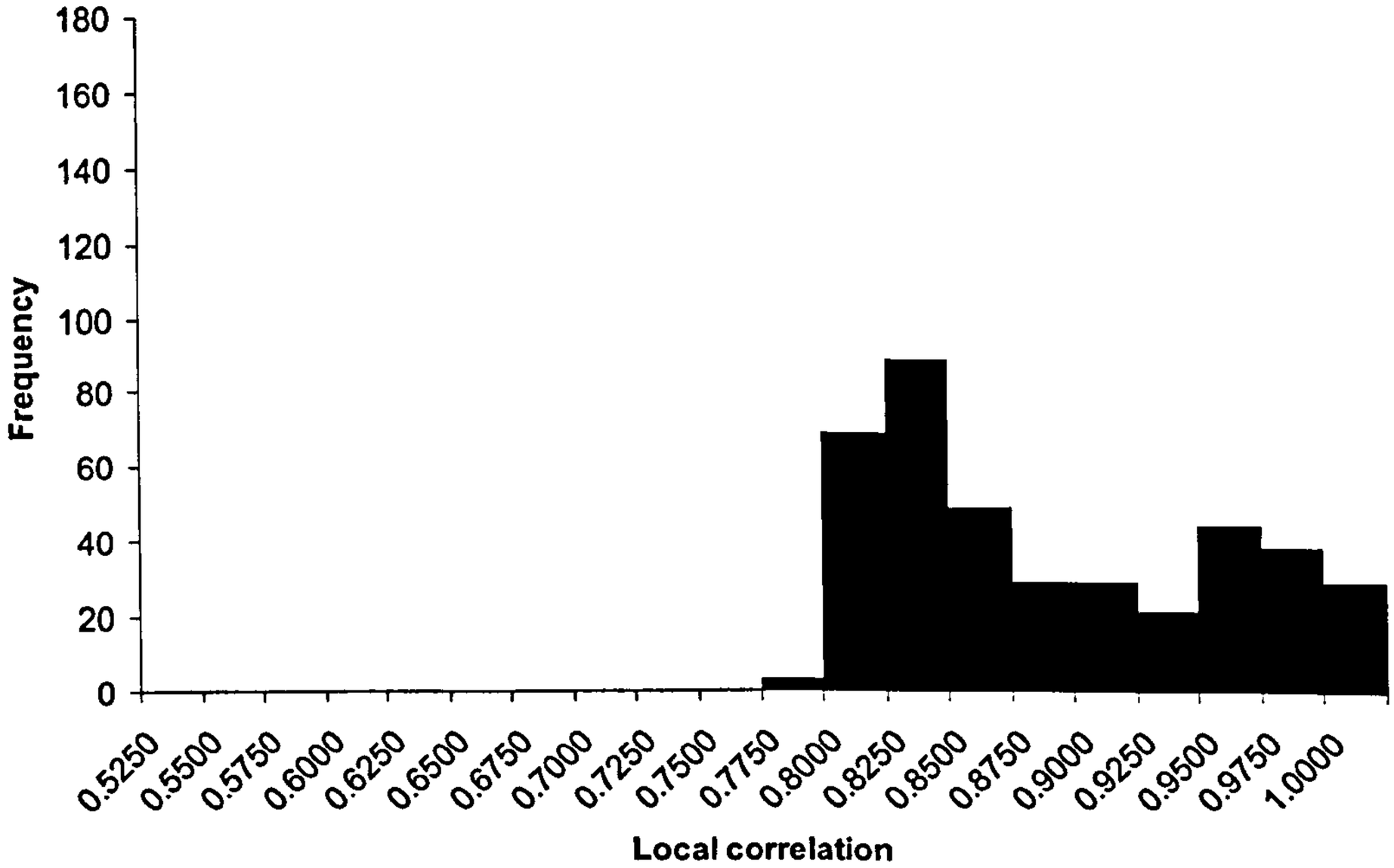


Figure 7.11: Frequency distribution of the cross-pattern local correlation values for character data at a neighbourhood distance, d=1.

Figure 7.12: Frequency distribution of the cross-pattern local correlation values for character data at a neighbourhood distance, d=2.

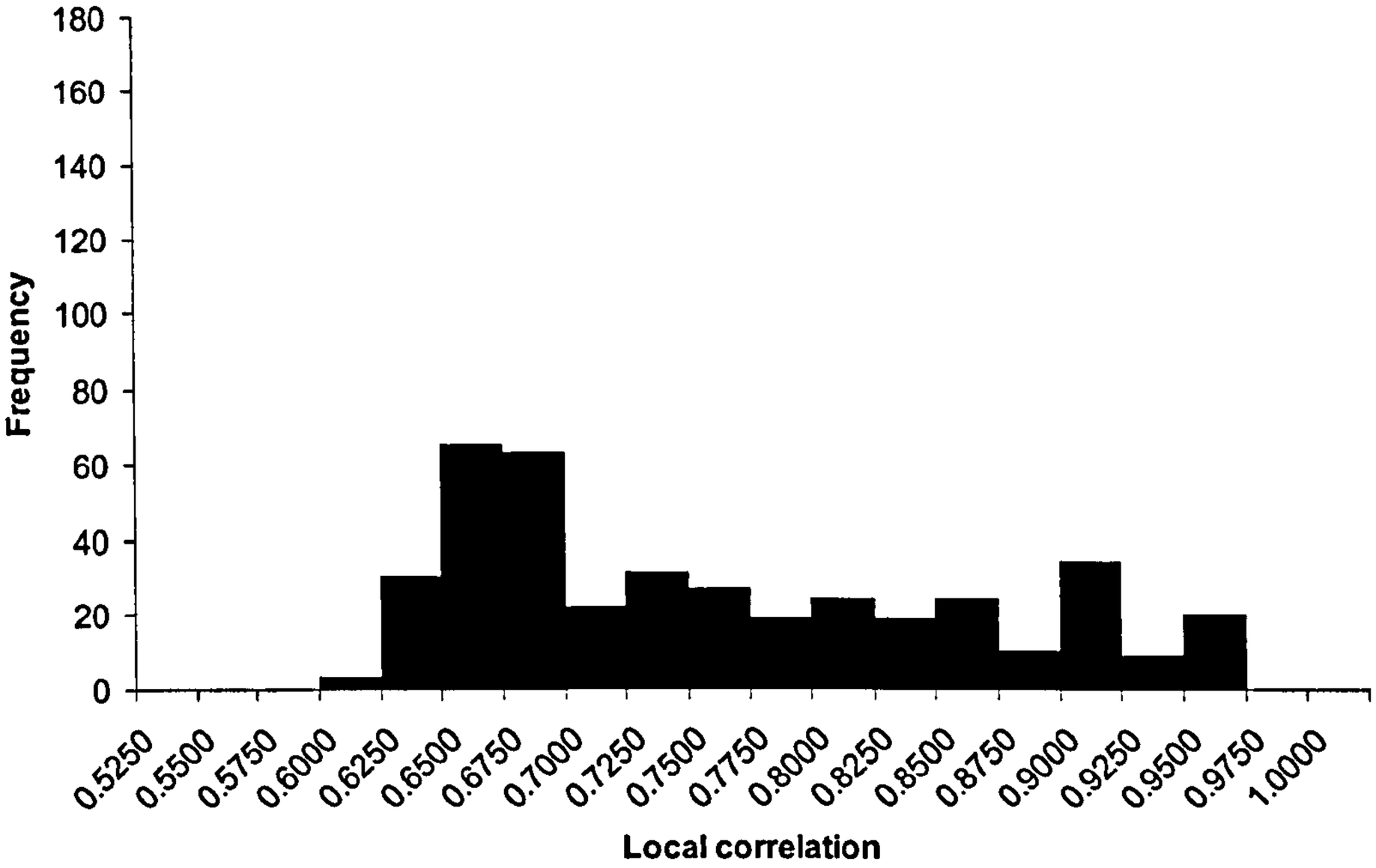


Figure 7.13: Frequency distribution of the cross-pattern local correlation values for character data at a neighbourhood distance, d=3.

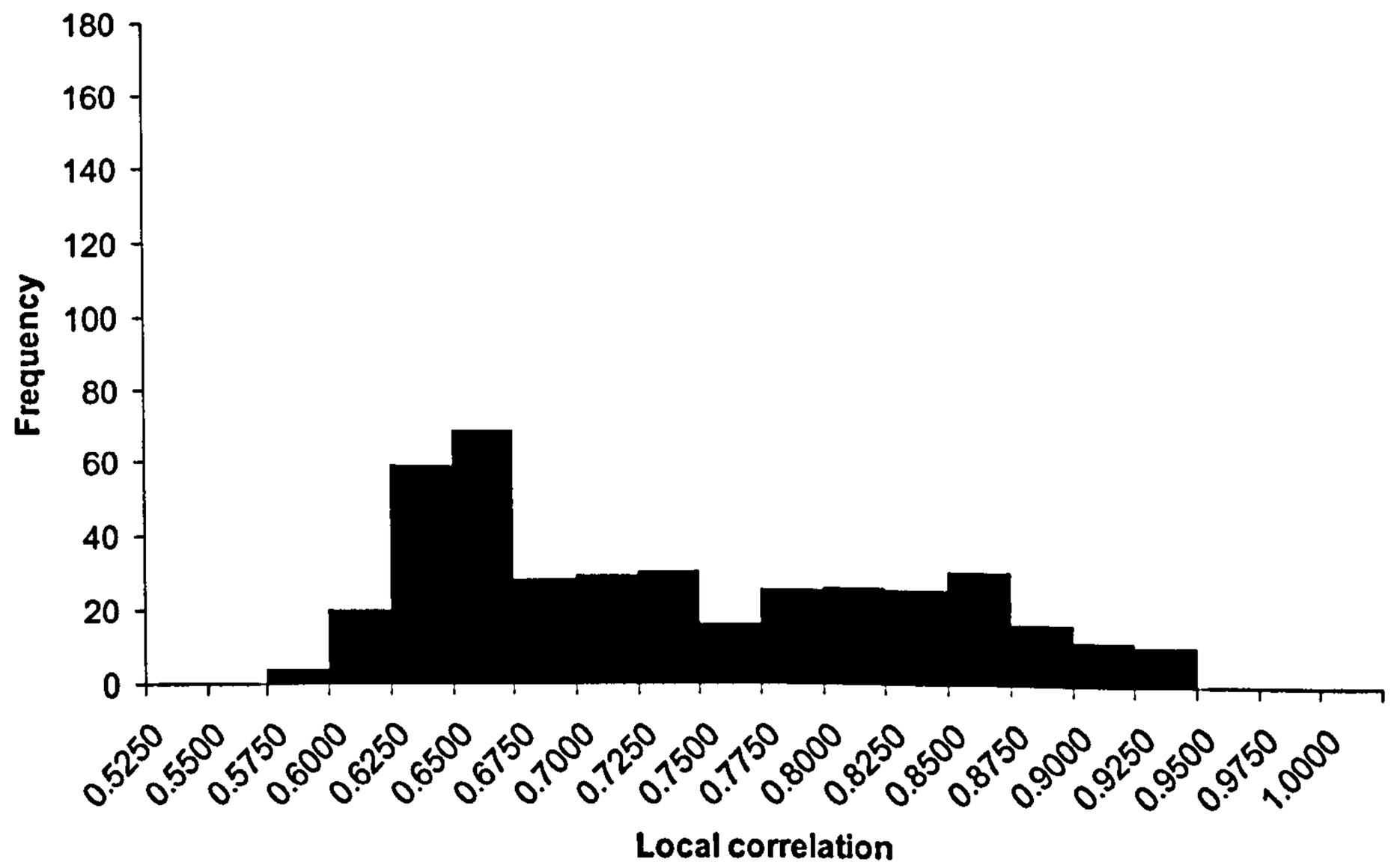


Figure 7.14: Frequency distribution of the cross-pattern local correlation values for character data at a neighbourhood distance, d=4.

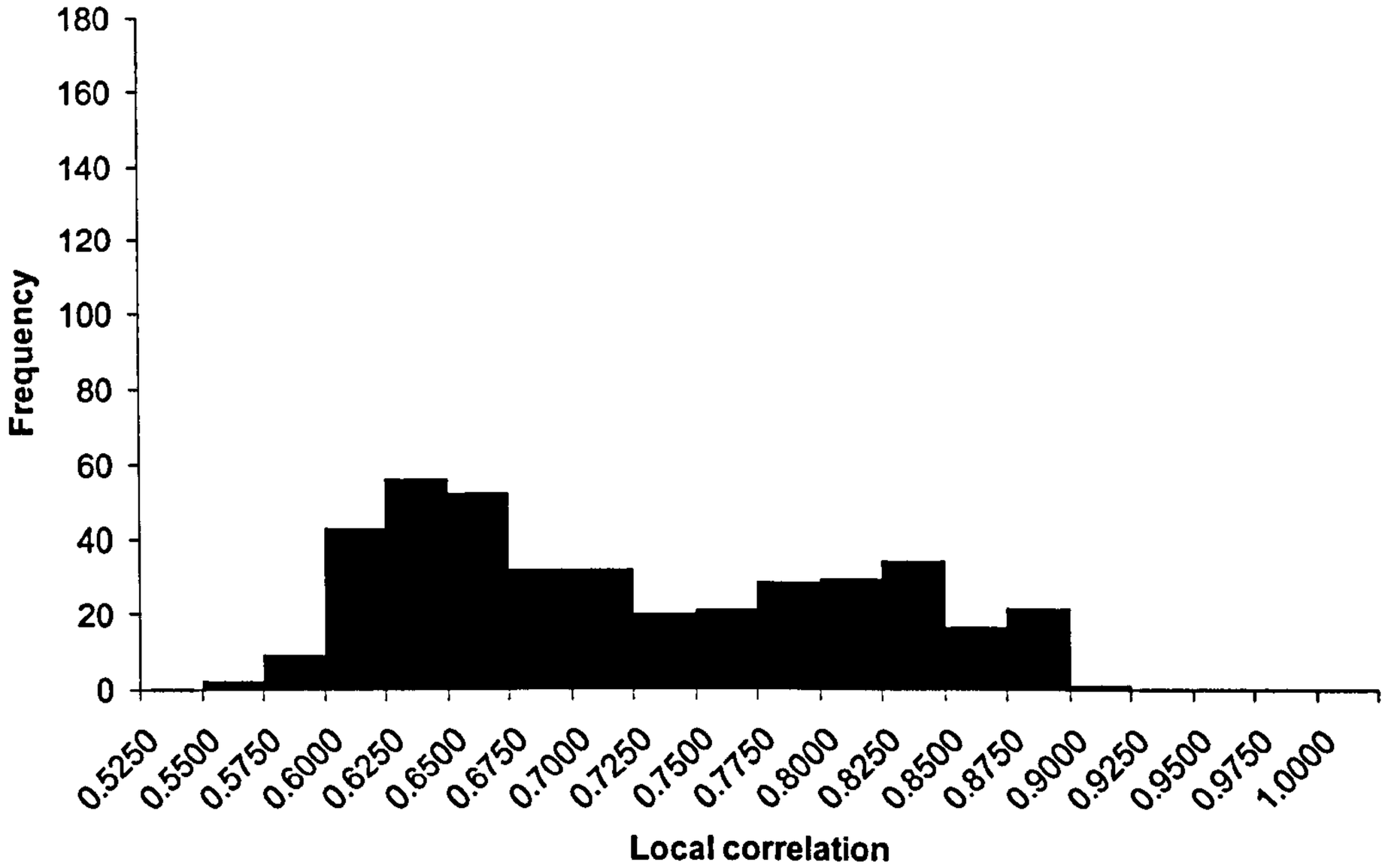


Figure 7.15: Frequency distribution of the cross-pattern local correlation values for character data at a neighbourhood distance, d=5.

The dominant feature of the preceding sequence of histograms is that the distributions are now, in contrast to those for geometric data, clearly non-Gaussian. The standard deviations of these distributions are also considerably larger.

The histograms show that, in contrast to those for geometric data, the only neighbourhood at which all bits exhibit a level of local correlation in excess of the global correlation is that at distance 1. As the neighbourhood size increases beyond a distance of 1 an increasing proportion of bits are correlated to a degree below the global level of 0.6918.

Re-examining the sample patterns shown in figure 7.2, it can be clearly seen that the patterns are made up of relatively thin lines rather than large blocks of black. The fact that the lines are thin has a direct impact on the range at which local correlation is present.

There is a cumulative effect introduced by this measure also. Each new neighbourhood includes, when calculating the local correlation for that neighbourhood, the correlation for the neighbourhoods within it. For example, the local correlation figure for a neighbourhood of size 4 includes within it the local correlation measures at neighbourhoods of sizes 1, 2, and 3. This effect could lead to a view that some degree of local correlation exists at a greater range

than is really the case. It is important therefore, to take this effect into account when considering the degree of local correlation present at any given neighbourhood size > 1.

Neighbourhood size	Mean local correlation
1	0.87 (s.d.=0.07)
2	0.78 (s.d.=0.10)
3	0.74 (s.d.=0.10)
4	0.71 (s.d.=0.10)
5	0.70 (s.d.=0.09)

Table 7.3: Mean local correlation values at various neighbourhood sizes for character training data.

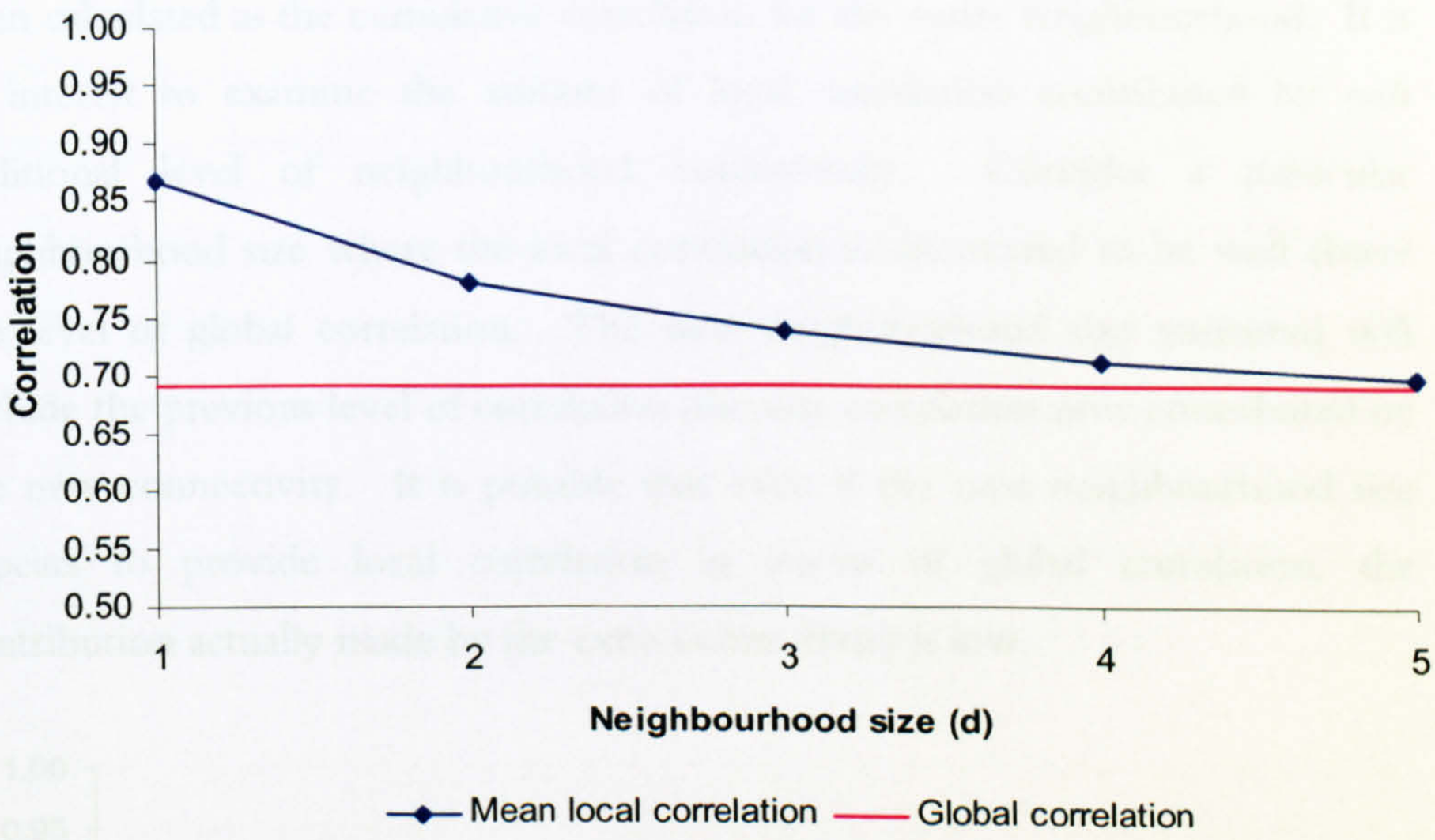


Figure 7.16: Mean local correlation against sub-pattern neighbourhood size for character training data. The level of global correlation is shown for comparison.

The first point of note from figure 7.16 (above) is how much higher the level of global correlation is for character training data than it was for the geometric data. This increase arises naturally from the increased level of bias across the character data set.

Secondly, at equivalent neighbourhood sizes, the level of local correlation is usually lower for character data than for geometric data though the difference is slight. A possible explanation for this is the fact that the geometric training patterns have large blocky areas of both black and white. This results in a large degree of local correlation over the entire pattern. In contrast, the character data has large areas of white (-1 bit) interrupted with thin black areas. As these black

areas are very thin, they are a) not locally correlated at any significant distance, and b) disrupting the local correlation of the white areas through which they pass.

The advantage in correlation gained by measuring sub-patterns of increasing size disappears far more quickly for character data than for geometric data. This can be seen by the near-convergence of the blue line, indicating local correlation, with the red. This is to be expected given the higher level of global correlation present in the character data.

7.4.2.3. Measuring Each Neighbourhood's Contribution to Correlation

Thus far, the level of local correlation measured at each neighbourhood size has been calculated as the cumulative correlation for the entire neighbourhood. It is of interest to examine the amount of local correlation contributed by *each* additional level of neighbourhood connectivity. Consider a particular neighbourhood size where the local correlation is discovered to be well above the level of global correlation. The next neighbourhood size measured will include the previous level of correlation plus any correlation now contributed by the new connectivity. It is possible that even if the next neighbourhood size appears to provide local correlation in excess of global correlation, the contribution actually made by the extra connectivity is low.

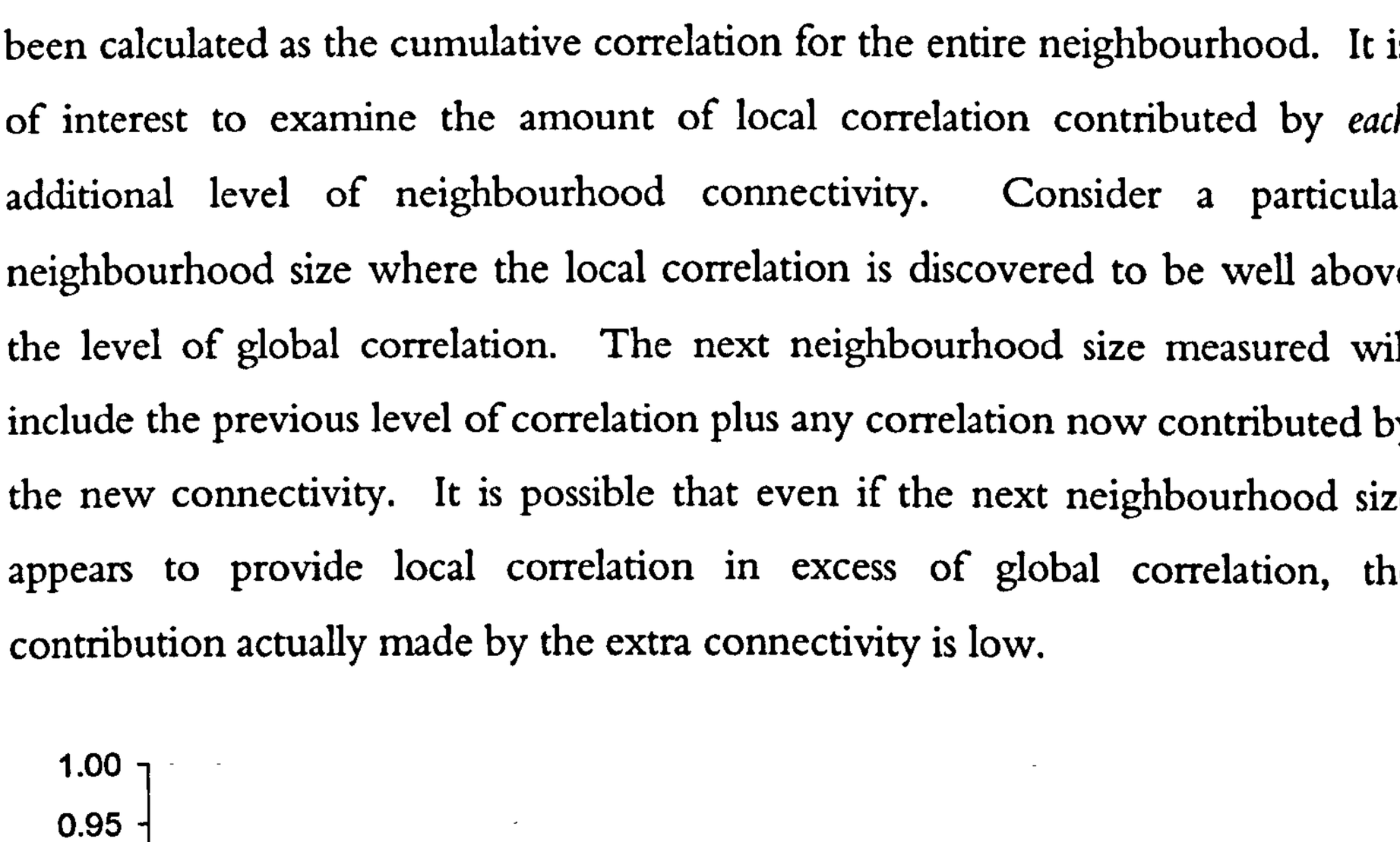


Figure 7.17: The level of local correlation introduced by each new level of neighbourhood connectivity for geometric and character data. The global correlation of the geometric and character data sets is indicated by the dotted and dot-dashed lines respectively.

Figure 7.17 illustrates, for both geometric and character data, the contribution to local correlation made by each neighbourhood. The dotted line indicates global correlation for the geometric data and the dot-dash line the same for character data.

It can be seen that for geometric data, larger neighbourhoods continue to contribute local correlation above the level of global correlation. The amount of correlation contributed falls linearly with the increasing neighbourhood size. This corresponds to the linear decline in the cumulative local correlation seen for geometric data in figure 7.10.

For character data, the last point at which a neighbourhood increase contributes local correlation above the global level is at d=3. Beyond this point larger neighbourhoods no longer contribute any greater benefit than would be obtained by simply choosing an equivalent number of bits to measure at random.

The implication of this measurement is that for geometric data, performance benefits should be seen when constructing networks with connectivity that reflects these neighbourhoods for all of the neighbourhood sizes examined here. For character data however, it might be expected that performance benefits would drop once neighbourhood connectivity had reached a size above d=3 due to the decline in the level of local correlation with bits at that range.

7.4.3. Measuring Site Activity within a Training Set

The following sequences of histograms show, for geometric data, character data, and their random data equivalents, the distribution of site activities for each training set. The mean values in each case should be equal to that of the overall level of pattern bias in the training set.

7.4.3.1. Geometric Data and Random Data (b=0.5)

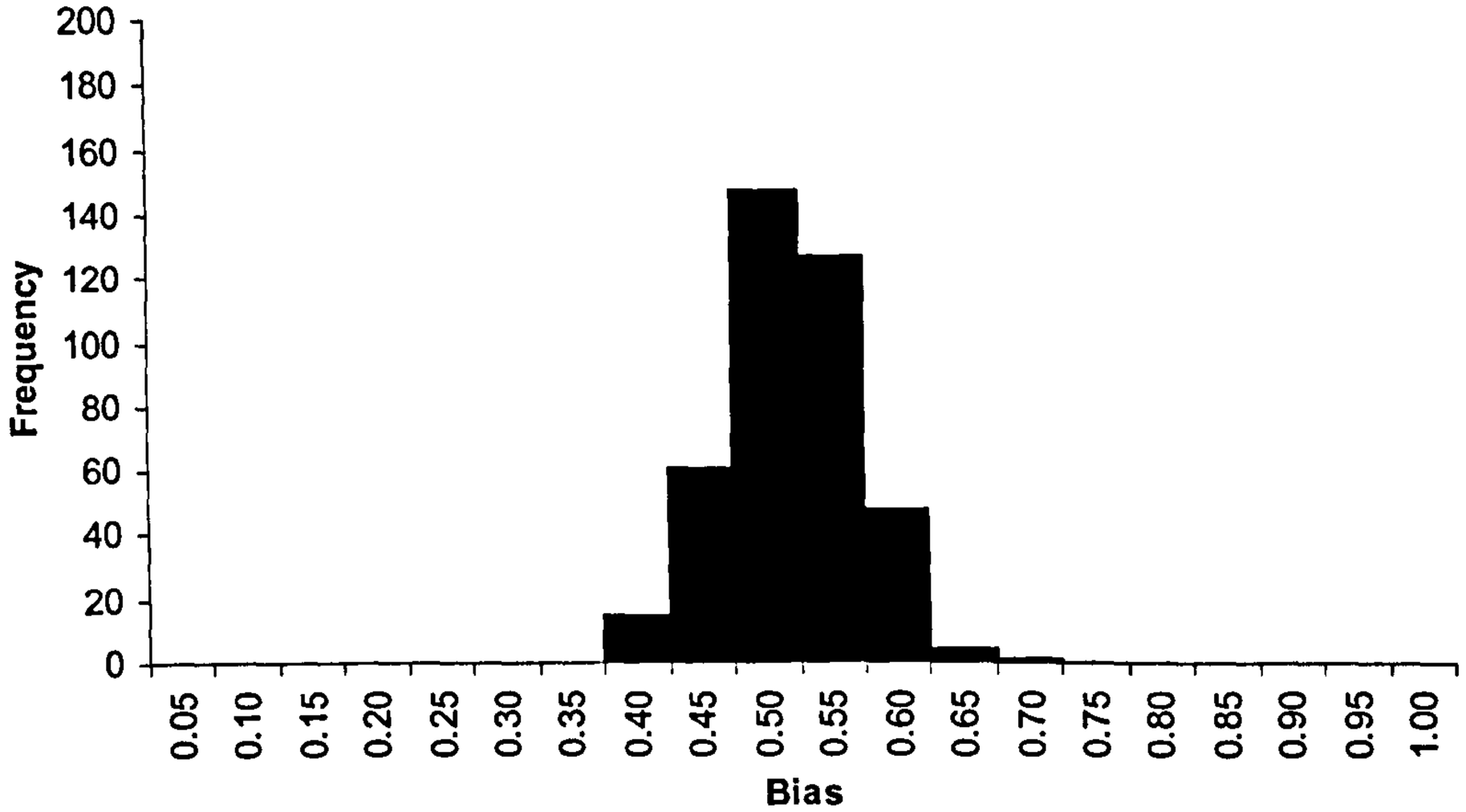


Figure 7.18: Frequency distribution of the site activity values for random data (b=0.5).

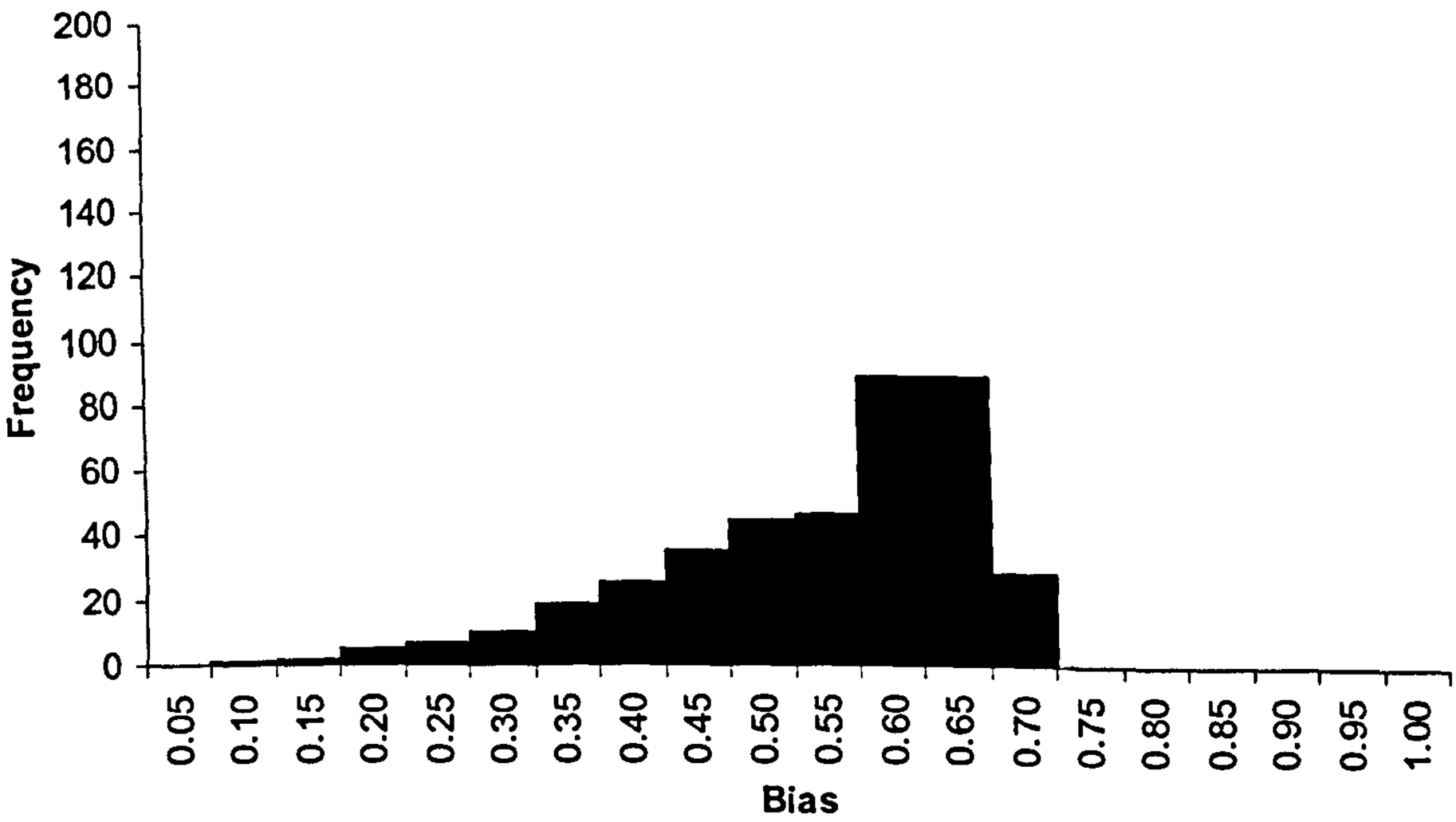


Figure 7.19: Frequency distribution of the site activity values for geometric data.

The 2 preceding histograms show the distribution of site activities for random unbiased data (figure 7.18) and geometric data (figure 7.19). As might be

expected, the values for random unbiased data are normally distributed with a mean value of 0.50. The range of values is from 0.36 to 0.66 and the standard deviation is 0.05.

The distribution of values for geometric data is negatively skewed though the mean value is still equal to that of the overall level of bias at 0.52. The range of values is from 0.10 to 0.70 and the standard deviation is 0.12.

From this it can be inferred that while the mean values are very similar and the training set bias, when calculated over all the patterns, is roughly 0.5 in each case, in the case of the geometric data an individual bit is more likely to have the same value throughout the training set when compared with the random data. For the geometric data, 32% of bits have a bias either lower than 0.4 or higher than 0.64. This means that a full 68% of bits have bias values that fall approximately within the entire range of values that were seen for the random data.

The above information shows that the set of data comprised of locally correlated patterns has a higher level of inter-pattern similarity, or correlation.

7.4.3.2. Character Data and Random Data (b=0.8)

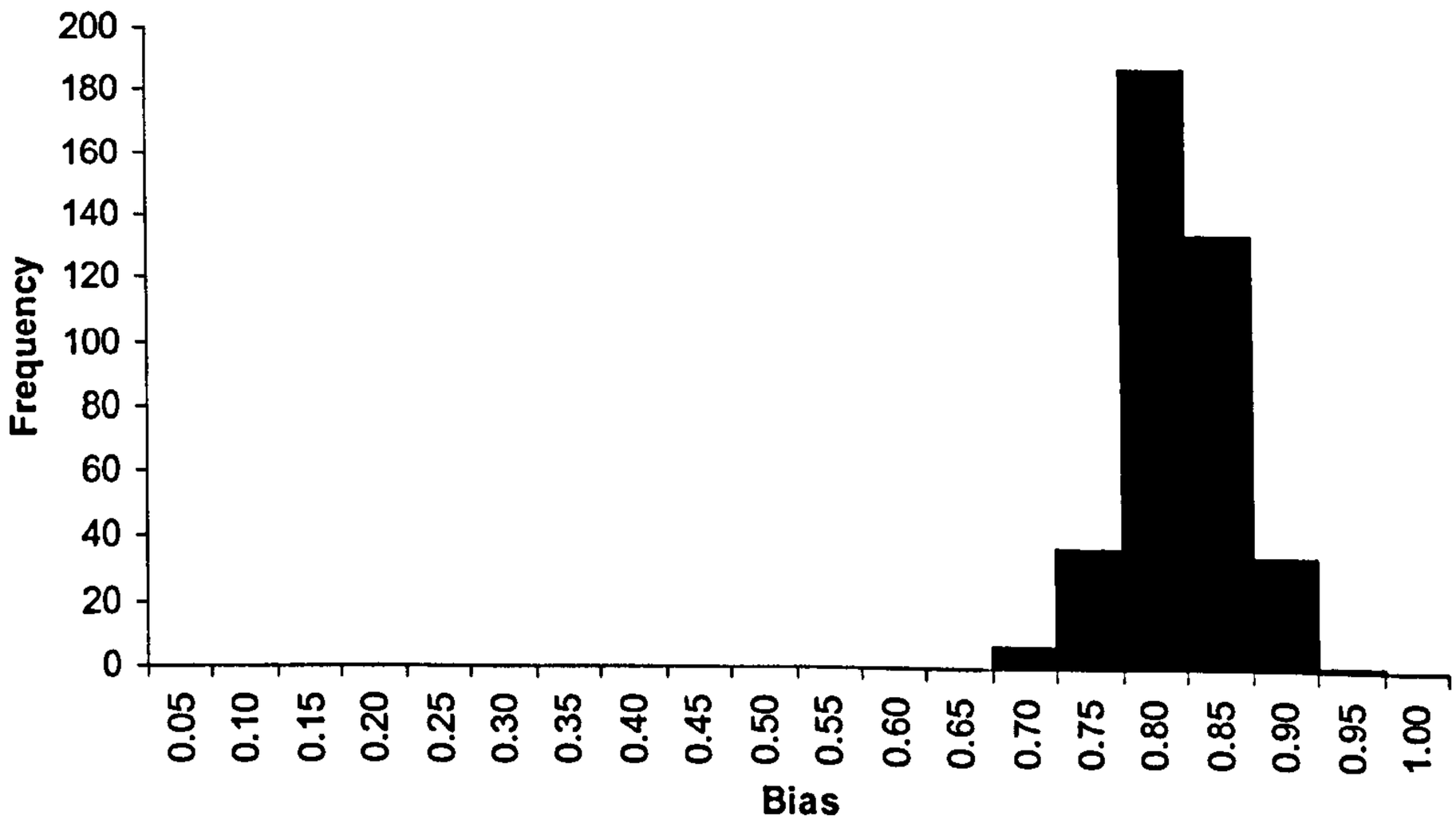


Figure 7.20: Frequency distribution of the site activity values for random data (b=0.8).

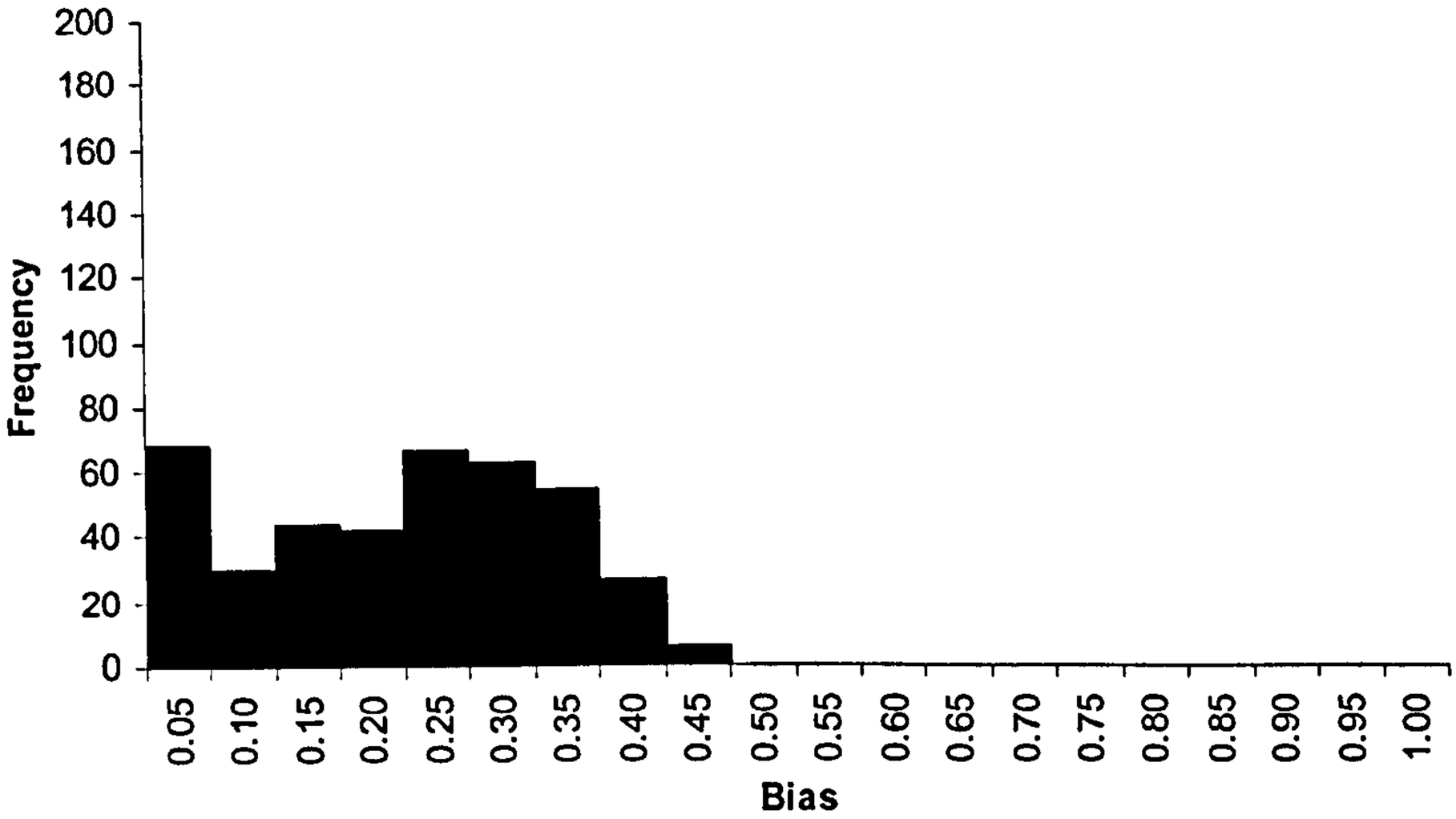


Figure 7.21: Frequency distribution of the site activity values for character data.

Figures 7.20 and 7.21 (above) show the distribution of site activities for random data with a bias of 0.8 and character data respectively. The values for the random data are again normally distributed with a mean value of 0.80. The range of values is from 0.68 to 0.91 and the standard deviation is 0.04.

The distribution of values for the character data is almost bimodal with an early peak in the frequency of bias and a remainder which looks approximately normal. The range of values is from 0.00 to 0.43 and the standard deviation is 0.12.

The slight bimodality of the distribution requires that more care is taken in interpreting the results. The initial peak in the distribution represents the fact that 17% of a network's neurons will be given the task of outputting a -1 at least 95% of the time. Close on one quarter of the neurons will be outputting -1 at least 90% of the time. It is clear to see that, for a large number of bits, the character data is very highly biased.

While this high degree of bias is evident, it is also the case that 53% of bits have a level of bias between the mean value, 0.20, and the upper end of the range of values, 0.43. The fact that just over one half of the bits have a level of bias below that which would be expected of random data with a similar level of overall bias means that the advantages of inter-pattern correlation may not be so evident with character data.

7.5. Discussion and Summary

The purpose of this section was to describe an artificially constructed set of training data, a set of analysis measures, and the results of their use.

Having initially identified some important requirements of the data, the manner in which the data was generated was described. The analysis tools were presented and descriptions of their use given. The design of these tools is to analyse, in a manner appropriate to this investigation, the underlying nature of the training data used in the remainder of this work.

The first analysis, which measured the bias of the training set, showed that the geometric and character data had levels of bias of 0.52 and 0.2 respectively. This indicates that the geometric data is practically unbiased while the character data possesses more than four times as many -1 valued bits as +1 ones.

The cross-pattern local correlation showed how the degree of correlation of a bit with those surrounding it varied according to how many of the surrounding bits were considered to be 'local' to it. This analysis showed that in the case of both geometric and character data, there was local correlation present that exceeded the level of correlation present in the data due to the pattern bias. The correlation for each locale tested was not markedly different when comparing between data types but the difference between local correlation and global correlation was greatest for geometric data. This arises from the higher level of global correlation already present due to the higher pattern bias of the character data.

Finally, the site activity analysis has showed that a considerable degree of inter-pattern similarity exists in both types of data. This similarity is more evident in the geometric data than in the character data and in the case of the latter is unlikely to present as much of an advantage as it is balanced by a larger degree of dissimilarity than was seen to be present in the equivalently biased random data.

In summary, this section has demonstrated that the natural data employed in this work possesses a level of local correlation that is greater than the respective level of global correlation. This, coupled with the inter-pattern correlation demonstrated using the site activity analysis, should, at least in the case of the geometric data, lead to an improvement in performance in networks where the connection topology is constructed at a similar range to that at which the local correlations exist. The investigation of this is reported in the next chapter.

8. ASSOCIATIVE MEMORY ARCHITECTURES WITH SPARSE CONNECTIVITY

8.1. Introduction

The aim of this series of experiments was to discover whether a topological bias towards local connectivity might permit greater performance than can be achieved using simple random connectivity when training using patterns exhibiting significant levels of local correlation. The possibility of this being a worthwhile area of investigation was mentioned by Canning and Gardner (1988) in their work on investigating the properties of partially connected neural networks.

There is a clear justification for why local connectivity might provide a performance improvement, especially in the area of capacity. Under the class 3, Gardner-type learning rules each neuron is trained as a perceptron (c.f. chapter 2). For individual perceptrons it is known that the maximum capacity *for unbiased random patterns* is equal to twice the number of incoming connections (Cover, 1965; Gardner, 1988). For a fully-connected network of perceptrons the theoretical maximum capacity is therefore $2(N)$ for N , where N is the size of the network. Furthermore, it was also shown by Gardner that the capacity will increase for patterns which are biased.

As described in chapter 5, Lopez and Schroder (1995) showed that it is not only the bias of the patterns that is important in increasing capacity but it is important that pairs or groups of correlated patterns have correlated outputs. In the case of the fully-connected networks this can be achieved by simply increasing the bias of the training patterns.

Consider training patterns of the types shown in §7.2. It was shown in the previous section that patterns of those types possessed a significant degree of local correlation on both an individual and an aggregate basis. High local correlation implies that individual bits often share the same value as their neighbours. If a bit in a particular position has the same value for a number of patterns in the training set then the correlation between the patterns should result in an environment in which Lopez and Schroder state that the associated perceptron should show improved capacity.

It is recognised that reducing the number of incoming connections to each perceptron will reduce the theoretical maximum capacity of each of them but it is hoped that this will be offset by the expected capacity improvements arising from the local correlations.

The performance of the networks is judged with respect to training time and capacity, connection storage efficiency, attractor performance, and the rate at which individual neurons failed with respect to increasing pattern load. In order to establish whether or not a performance improvement arises from using locally correlated data with local connectivity, networks with different styles of connectivity were trained using data within which local correlation is present, and data within which it is not or is very low.

This chapter considers a series of evaluative experiments measuring the performance of the networks. The critical result presented is that of *useful capacity*, a term described in detail in §8.4.2.

8.2. Network Architecture, Learning Rule, and Training Data

The networks employed in this series of experiments were of size N=400 and the neurons are considered to be arranged as if on a 20-by-20 grid. Connectivity between neurons was established through the use of one of two strategies described below.

The networks' weight matrices were generated using the Symmetric Local Learning algorithm described in §2.2.

Two categories of training data were used over the course of these experiments.

a) *Artificially generated non-random data.* This data was generated as described in chapter 7. Two different types of data were employed; data derived from computer character sets and data artificially generated using random placement of small geometric shapes within the 2-dimensional training pattern representation.

b) *Randomly generated data.* This was generated as required with levels of bias 0.5 and 0.8. These levels of bias are very close to that present within the geometric shape data and character data summarised above (c.f. chapter 7).

8.3. Network Performance Analysis

The performance of the trained networks was assessed according to four criteria: training time, capacity, the ability of the stored patterns to act as attractors, and the rate at which the number of neurons which fail to train rises as pattern load increases.

Training time is taken to be the number of presentations of the training set required to successfully train the network. The upper bound of the number of presentations, beyond which a network is deemed to have failed to train, is 1000. This value was chosen to be well in excess of that ordinarily required to train fully-connected networks with the same learning rule (c.f. chapter 4).

Storage efficiency is defined as being the ratio of the number of successfully stored patterns to the mean number of incoming connections to each neuron.

Attractor performance was assessed using the modified version of the measure developed by Kanter & Sompolinsky (1987) and described in detail in chapter 3.

Neuron failure rate is taken as a count of the neurons that fail to achieve an aligned local field that exceeds the training threshold within some prescribed number of iterations as load is increased.

8.4. Establishing Connectivity

To establish whether or not the structure of the networks' connectivity has any significant effect on the performance characteristics of the network, two connection strategies were employed. These strategies are termed *random connectivity* and *nearest neighbour connectivity* and are described in detail below.

8.4.1. Random Connectivity

Random connectivity is established as follows: The number of connections that would be present if the network were to be fully-connected is calculated as

$N * (N - 1)$. A proportion of this value is taken to be the desired level of connectivity within the new network. This figure is divided by two to give the total number of connection pairs required to achieve this level of connectivity. Connection pairs are specified in order that symmetry within the weight matrix may be maintained and so the presence of simple update dynamics may be relied upon (c.f. §2.3).

Having established a figure for the number of connection pairs required, pairs of neurons are selected at random. Once selected, should a connection not already be present, a bi-directional connection is created between them. This process continues until the specified level of connectivity has been reached.

Example

Number of neurons (N)	= 400
Number of connections in fully connected network	= 159,600
Required level of connectivity	= 0.5
Connection pairs in sparse network	= (0.5 * 159,600) / 2
	= 39,900

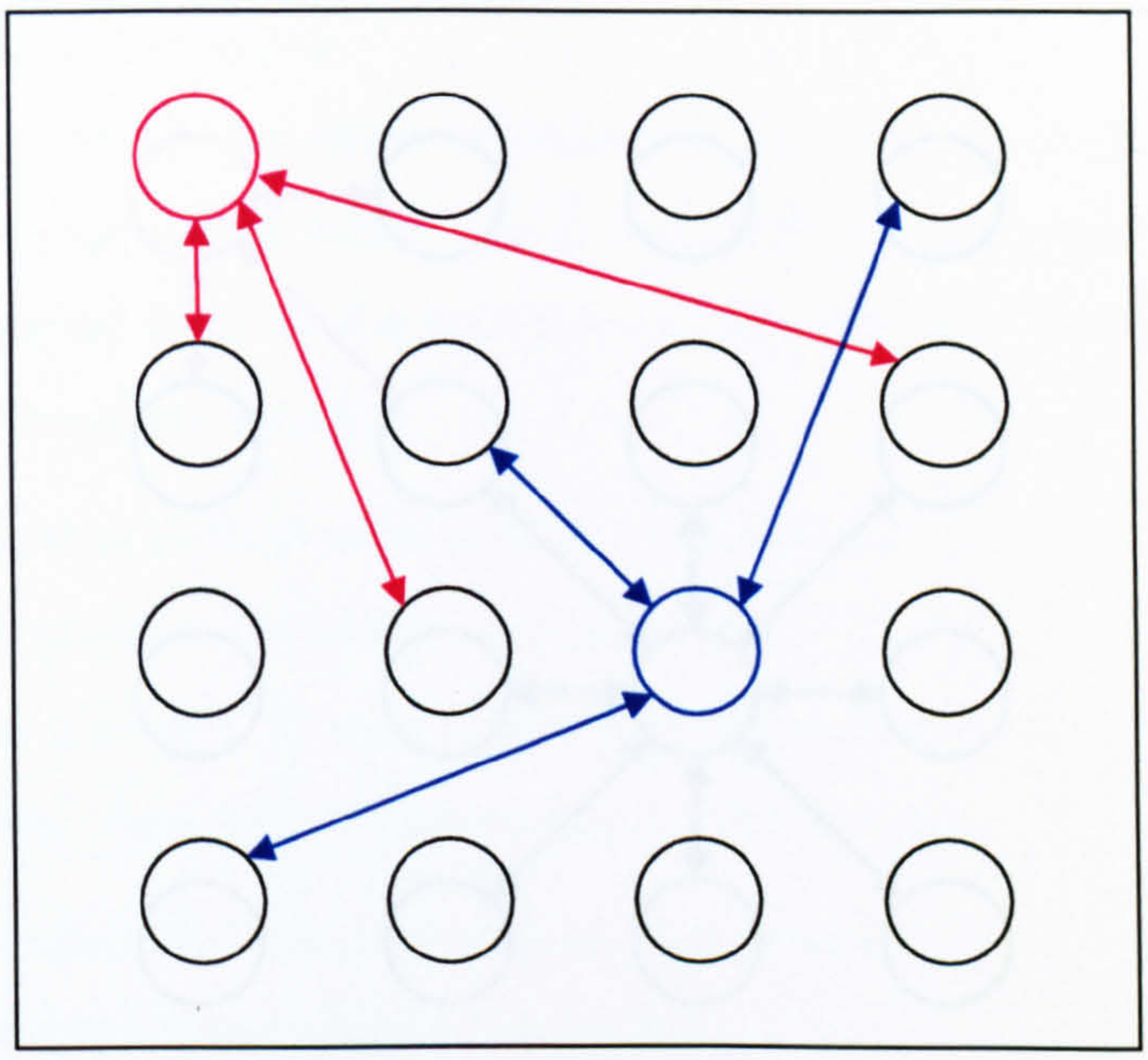


Figure 8.1: A pictorial representation of a small network within which random connectivity has been established. Connections are shown for two neurons as an example.

8.4.2. Nearest Neighbour Connectivity

The generation of neighbourhood-based connectivity is carried out in a slightly different manner to random connectivity, above. The neurons are arranged in a conceptual grid corresponding to pixel positions in pictorial data. A distance, *d*, for the neighbourhood is chosen in the same manner as used for calculating local correlation in training patterns (c.f. chapter 7). Next, network neurons are taken in sequence from the top left of the grid and an incoming connection is established to the current neuron from all neurons at or closer than *d* neurons away. Here, unidirectional connections are created as symmetry arises naturally from the creation of the connectivity of subsequent neurons.

Connectivity does not wrap-around at the edges of the grid. This requirement ensures that the structure of the connectivity reflects the way in which the local correlation within the training patterns is calculated.

Neurons with a reduced neighbourhood i.e. those at the edges and corners of the network, will have a lower level of connectivity than those which possess a full neighbourhood. This further reduction in connectivity could cause those neurons affected to fail much sooner than those with larger neighbourhoods or neurons with random connectivity.

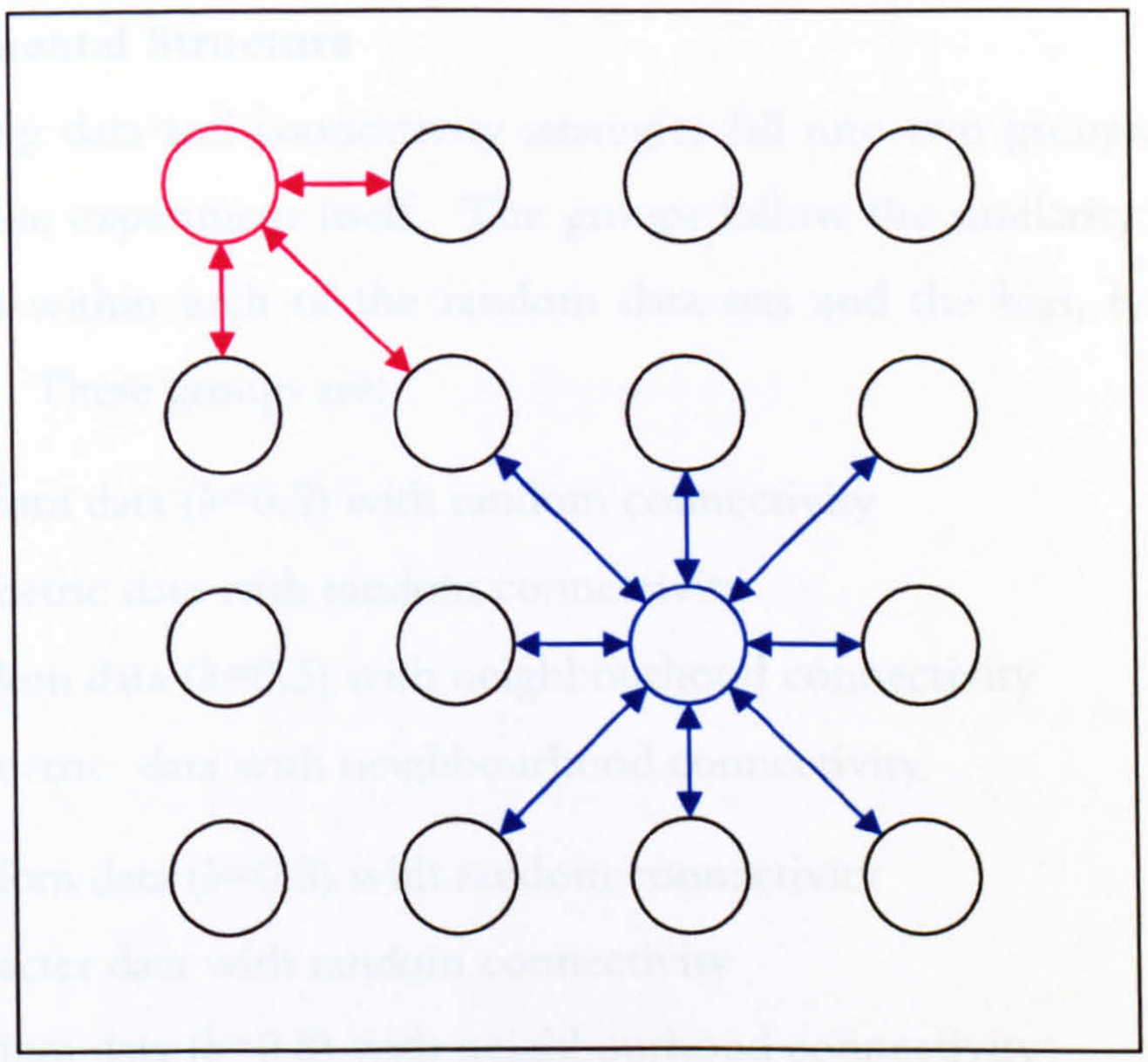


Figure 8.2: A pictorial representation of a small network within which neighbourhood connectivity has been established at a distance (d) of 1. Connections are shown for two neurons as an example.

Local connectivity changes the appearance of the training set from the perspective of individual neurons (Karlholm, 1993). Critically, the locally correlated nature of the training data means that neurons will have input patterns that are individually both highly biased and correlated with the associated output. The idea is that when an individual pixel in any pattern is on, there is a good chance that many of the surrounding pixels will also be on due to the spatial continuity of real images. Local connectivity should therefore, according to the work of Lopez and Schroder (1995), lead to increased capacity for a large number of neurons and which should, in turn, lead to a relatively low number of neurons failing to train.

The presence of a reduced number of failed neurons assists in the definition of the term *useful capacity*. In later work, the effect of adding additional connections after attempting a first phase of training was investigated. The additional connections are added in order to try and assist failed neurons in successfully learning their input patterns. High useful capacity occurs for networks where only a small number of failed neurons occur as the loading rises past the point at which all the training patterns are stable. The small numbers of failed neurons allow the networks to be compensated with extra connectivity at low cost in terms of additional connections.

8.5. Experimental Structure

As the training data and connectivity strategies fall into two groups, so does the structure of the experiment itself. The groups follow the similarity between the levels of bias within each of the random data sets and the bias, b, of the non-random data. These groups are:

a) Random data (b=0.5) with random connectivity

   Geometric data with random connectivity

   Random data (b=0.5) with neighbourhood connectivity

   Geometric data with neighbourhood connectivity

b) Random data (b=0.8) with random connectivity

   Character data with random connectivity

   Random data (b=0.8) with neighbourhood connectivity

   Character data with neighbourhood connectivity

In each group there are networks with random connectivity and neighbourhood connectivity. Neighbourhood sizes of 1 to 5 were employed. The random connectivity was created such that the overall level of connectivity as a proportion of the number of connections present if the network were fully-connected would be equal to the levels of connectivity using neighbourhoods. These equivalences are as follows:

Neighbourhood distance (d)	Connectivity	Mean connections per neuron
1	0.0185	7.41
2	0.0527	21.09
3	0.0999	39.96
4	0.1575	63
5	0.2231	89.25

Table 8.1: Connectivity level equivalences between connectivity established by random means and that established using neighbourhood connectivity. Also shown is the corresponding mean number of connection at each neuron for each level of connectivity.

The last column in the table above, the number of mean connections per neuron (MCPN), provides a method of referring to the level of connectivity in any given network without the need for specifying the nature of the pattern of connectivity.

It is important to note that neurons with a reduced neighbourhood, i.e. those at the edges and corners of the network, will have a lower level of connectivity than the mean stated in the table above. As mentioned earlier, it is this further reduction in connectivity that may cause those neurons affected to fail much sooner than those with larger neighbourhoods or neurons with random connectivity.

8.6. Results

It is important to note that these results require careful interpretation in the context of the known issues regarding the likely early failure of corner and edge neurons in the case of neighbourhood connectivity.

It is expected that capacity and attractor performance will often appear to be worse for networks with neighbourhood connectivity than for those with random connectivity due to this early neuron failure.

As the capacity finds the first point at which at least one neuron fails to train it is a fallible guide to actual capacity in these networks with low levels of connectivity. Low connectivity may lead to high variability in the observed capacity. A more meaningful measure of capacity, useful capacity, was given in §8.4.2.

Moreover, for the networks with neighbourhood connectivity, the edge and particularly corner neurons will grossly distort this measured capacity as they are very likely to fail quickly. For example, at a neighbourhood size of d=1, corner neurons only have 3 inputs.

The important results in this chapter are those detailing the number of failed neurons at various degrees of pattern load, as it is networks with small numbers of failed neurons that may be compensated cheaply with extra connectivity (c.f. chapter 9).

8.6.1. Capacity and Training Time

Capacity and training time results are presented as summary tables for conciseness. The full results tables from which the summaries have been produced may be found in appendix D. Each value used in creating the summary represents is averaged over five simulation runs.

8.6.1.1. *Results for Random (b=0.5) and Geometric Data*

Tables 8.2 and 8.3 show the capacity and training time result summaries for networks learning random data (bias 0.5) and geometric data. Results for each of the levels of connectivity described in table 8.1 are shown. The networks are compared in two ways: firstly, the effect that the type of training data has on capacity and training time is examined (figure 8.2). The training data type resulting in the highest capacity and shortest training time is given for each of the two connectivity types.

Secondly, the effect of the connectivity strategy is examined with respect to the two types of training data (figure 8.3). The connectivity strategy resulting in the highest capacity and shortest training time is given for each of the two data types.

Comparing between random data (b=0.5) and geometric data

MCPN		Random connectivity	Neighbourhood connectivity
7.41	**Capacity**	Failed to train	Failed to train
	Training time	Failed to train	Failed to train
21.09	**Capacity**	Higher for random data	Higher for random data
	Training time	Inconclusive	Inconclusive
39.96	**Capacity**	Higher for geometric data	Higher for geometric data
	Training time	Shorter for random data	Shorter for random data
63	**Capacity**	Higher for geometric data	Higher for random data
	Training time	Shorter for random data	Shorter for random data
89.25	**Capacity**	Higher for random data	Higher for random data
	Training time	Shorter for random data at low loadings. Shorter for geometric data at higher loadings.	Shorter for random data

Table 8.2: Results of capacity and training time comparisons between random (b=0.5) and geometric data types at each of five levels of random or neighbourhood connectivity. The type of data resulting in the highest capacity or lowest training time is given for each case.

Table 8.2 shows that network capacity is usually higher when learning random data rather than geometric data regardless of the connectivity strategy used. This is the case for levels of connectivity of 21.09 and 89.25 MCPN. At 39.96 MCPN the capacity is higher for geometric data on both types of connectivity. An interesting point of note occurs at 63 MCPN where the capacity is higher for geometric data on random connectivity and higher for random data on neighbourhood connectivity.

The results for training time from table 8.2 indicate that where numerical results exist, random data almost always trains more quickly than geometric data. The exception to this is at 89.25 MCPN where the training time is shorter for random connectivity at low loadings but becomes longer as the number of patterns on which the network is trained increases (see figure 8.3). This is contrary to what might be intuitively expected from the network. If local connectivity is to provide some benefit in performance it would be expected that both types of data would produce similar results on random connectivity. Effects such as this and potential causes and reasons for their existence are discussed at the end of this chapter.

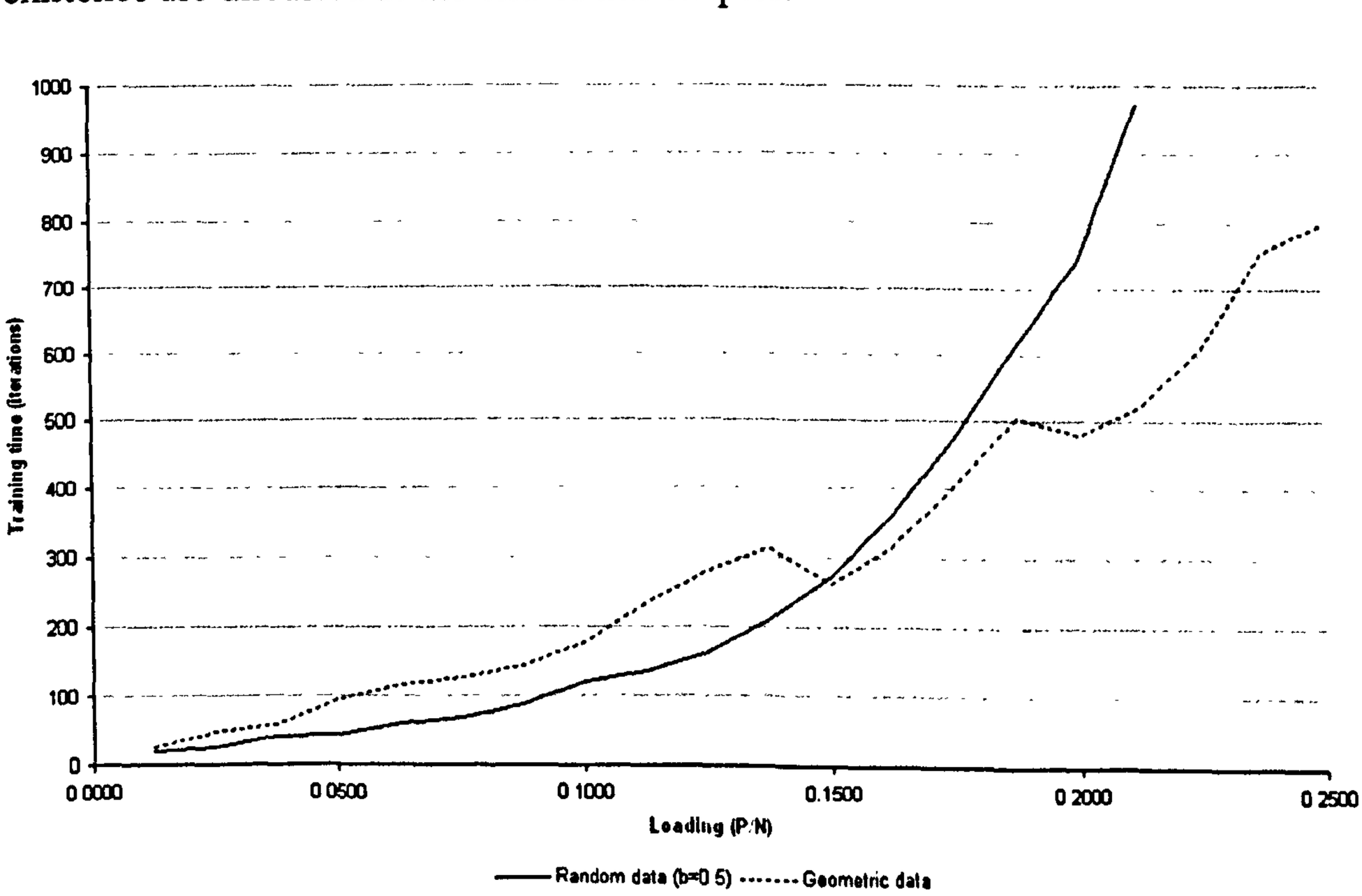


Figure 8.3: Training time against pattern load for networks with random connectivity learning random (b=0.5) and geometric data. Training time is shorter for random data (solid line) at low loadings (< 0.1500) but shorter for geometric data (dashed line) at higher loadings.

Table 8.3 (below) shows that, at every level of connectivity for which numerical results exist, the capacity of networks created with random

connectivity is greater than or equal to those where connectivity was establish using the neighbourhood method regardless of the data type being learnt.

The training time results are not what one might expect given the capacity summaries. Neighbourhood connectivity results in a shorter training time when using either data type at 21.09 or 39.96 MCPN but training time is shorter for random connectivity at 63 and 89.25 MCPN.

For training time, at 39.96 MCPN a similar effect as was described using figure 8.3 is observed. Although neighbourhood connectivity eventually results in shorter training time with geometric data, at low loadings the training times are faster for random data.

Comparing between random connectivity and neighbourhood connectivity

MCPN		Random (b=0.5) data	Geometric data
7.41	**Capacity**	Failed to train	Failed to train
	Training time	Failed to train	Failed to train
21.09	**Capacity**	Same for both strategies	Higher for random connectivity
	Training time	Shorter for neighbourhood connectivity	Shorter for neighbourhood connectivity
39.96	**Capacity**	Higher for random connectivity	Higher for random connectivity
	Training time	Shorter for neighbourhood connectivity	Shorter for random connectivity at low loadings. Shorter for neighbourhood connectivity at higher loadings
63	**Capacity**	Higher for random connectivity	Higher for random connectivity
	Training time	Shorter for random connectivity	Shorter for random connectivity at low loadings. Inconclusive at higher loadings
89.25	**Capacity**	Higher for random connectivity	Higher for random connectivity
	Training time	Shorter for random connectivity	Shorter for random connectivity

Table 8.3: Results of capacity and training time comparisons between random and neighbourhood connectivity strategies at each of five levels of connectivity for networks learning random data (b=0.5) and geometric data. The pattern of connectivity resulting in the highest capacity or shortest training time is given for each case.

8.6.1.2. Results for Random (b=0.8) and Character Data

Table 8.4 shows, in the same manner as for random (bias 0.5) and geometric data, capacity and training time results for random data with bias 0.8 and character data.

Comparing between random data (*b*=0.8) and character data

MCPN		Random connectivity	Neighbourhood connectivity
7.41	**Capacity**	Failed to train	Failed to train
	Training time	Failed to train	Failed to train
21.09	**Capacity**	Higher for random data	Higher for random data
	Training time	Inconclusive	Inconclusive
39.96	**Capacity**	Higher for random data	Same for both data types
	Training time	Shorter for random data	Shorter for character data
63	**Capacity**	Higher for random data	Same for both data types
	Training time	Shorter for random data	Shorter for random data at low loadings. Shorter for character data at higher loadings
89.25	**Capacity**	Higher for random data	Higher for random data
	Training time	Shorter for random data	Shorter for random data

Table 8.4: Results of capacity and training time comparisons between random (*b*=0.8) and character data at each of five levels of random or neighbourhood connectivity. The type of data resulting in the highest capacity or shortest training time is given for each case.

Table 8.4 shows that, for all levels of connectivity at which numerical results exist, the capacity of networks learning random (*b*=0.5) data is greater than equal to that of those learning character data regardless of the connectivity strategy used.

The training time results show that for 39.96 and 63 MCPN, the training time is shorter for random data on random connectivity but it is shorter for character data on neighbourhood connectivity. At 63 MCPN however, this latter result only becomes true at higher pattern loadings. At 89.25 MCPN it would appear that the larger neighbourhood has lost any advantage that may

have been present at lower levels of connectivity and random data again results in the quickest training time.

Comparing between random connectivity and neighbourhood connectivity

MCPN		Random data (b=0.8)	Character data
7.41	**Capacity**	Failed to train	Failed to train
	Training time	Failed to train	Failed to train
21.09	**Capacity**	Same for both strategies	Failed to train
	Training time	Shorter for random connectivity	Failed to train
39.96	**Capacity**	Higher for random connectivity	Higher for neighbourhood connectivity
	Training time	Shorter for random connectivity	Shorter for neighbourhood connectivity
63	**Capacity**	Higher for random connectivity	Higher for random connectivity
	Training time	Shorter for random connectivity	Shorter for neighbourhood connectivity
89.25	**Capacity**	Higher for random connectivity	Higher for random connectivity
	Training time	Shorter for random connectivity	Shorter for neighbourhood connectivity

Table 8.5: Results of capacity and training time comparisons between random and neighbourhood connectivity strategies at each of five levels of connectivity for networks learning random data (b=0.8) and character data. The pattern of connectivity resulting in the highest capacity or shortest training time is given for each case.

Table 8.5 shows that, in capacity terms, random connectivity generally appears to be a more advantageous connectivity strategy than neighbourhood connectivity whether the data being learnt is the random (b=0.8) set or the character data. This can be seen in the summary of capacity results for levels of connectivity of 63 and 89.25 MCPN. A different result exists for 39.96 MCPN. At this level of connectivity capacity is higher for random connectivity when learning random data and higher for neighbourhood connectivity when learning character data. Given the fact that when learning character data, the lowest level of connectivity at which the networks where capable of being successfully trained was 39.96 MCPN, this would seem to indicate that, at least in this case where the data is highly biased, as small a neighbourhood as is possible is preferable when learning locally correlated data.

The results for training time show a clear correlation between the type of data being learnt and the connectivity strategy resulting in the shortest training time. The networks only produced enough results for comparison at loadings of 39.96, 63, and 89.25 MCPN. At these loadings, the training time is shorter for random connectivity when learning random data and shorter for neighbourhood connectivity when learning character data.

8.6.2. Storage Efficiency

Storage efficiency is taken to be the ratio of the maximum number of patterns successfully stored to the mean number of connections per neuron. This value should provide an indication of how efficient a particular level of connectivity is when learning a certain type of training data. The results used are the same as those for the previous section on training time and capacity. Therefore, the capacity values used are once again the mean of five simulation runs.

8.6.2.1. Results for Random (b=0.5) and Geometric Data

	Random connectivity		**Neighbourhood connectivity**	
MCPN	**Random data**	**Geometric data**	**Random data**	**Geometric data**
7.41	Failed to train	Failed to train	Failed to train	Failed to train
21.09	0.71	0.47	**0.71**	0.23
39.96	0.75	0.88	0.25	**0.50**
63	0.87	1.11	0.56	0.32
89.25	**0.95**	**1.12**	0.50	0.28

Table 8.6: Storage efficiency values calculated as the ratio of the number of successfully trained patterns to the mean number of connections per neuron. Values highlighted with bold text are the maximum value for each data/connectivity type pairing.

Table 8.6 shows the following: For each network (defined by the pairing of a set of training data and connectivity type) the storage efficiency is shown for a number of levels of connectivity (mean number of connections per neuron). The peak efficiency for each network is highlighted using bold text. As with the training time results, no values are present for networks with 7.41 MCPN. No network managed to store the minimum of five patterns with such a low level of connectivity.

It can be seen from the table that, for networks with random connectivity, the efficiency increases with the level of connectivity. However, for networks

employing neighbourhood connectivity and learning geometric data there is a clear peak in efficiency at a connectivity level of 39.96 MCPN. From table 8.1 it can be seen that this level of neighbourhood connectivity is equivalent to a neighbourhood size, *d*, of 3.

As stated in §2.4.1, it is known (Gardner, 1988) that the maximum capacity for a network being trained on uncorrelated patterns with Gardner class rules is $2N$, where N is the input dimensionality of the neurons. It is clear from the data that none of the networks are even close to that figure. There are a number of reasons why this might be the case. The first of these is that the theoretical maximum capacity figure of $2N$ will only be seen in the limit N (Gardner, Gutfreund et al., 1989). It is therefore reasonable to expect that the networks being examined in this work will perform below this maximum value.

A compounding factor is that the connectivity level quoted is the *mean* number of connections per neuron. Due to the pattern of connections for the networks employing neighbourhood connectivity some neurons, namely those at the corners and edges, will have substantially less connections than the mean figure might suggest. As it is only required that one neuron fail for the entire network to be considered to have failed to train it is likely that these edge and corner neurons are at least partially responsible for lower storage efficiency figures than might be expected.

8.6.2.2. Results for Random (b=0.8) and Character Data

	Random connectivity		Neighbourhood connectivity	
Connectivity	**Random data**	**Character data**	**Random data**	**Character data**
7.41	Failed to train	Failed to train	Failed to train	Failed to train
21.09	0.24	Failed to train	0.24	Failed to train
39.96	0.90	0.38	0.50	**0.50**
63	1.03	**0.71**	0.48	0.48
89.25	**1.06**	0.67	**0.73**	**0.50**

Table 8.7: Storage efficiency values calculated as the ratio of the number of successfully trained patterns to the mean number of connections per neuron. Values highlighted with bold text are the maximum value for each data/connectivity type pairing.

Table 8.7 shows that, as was the case for unbiased random and geometric data, the storage efficiency tends to increase with the level of connectivity for networks with random connectivity trained using random data. The storage efficiency for these networks is higher for random data than for geometric data at equivalent loadings and while it is difficult to say for certain, it would appear that for both data types any significant improvement in storage efficiency disappears above 63 MCPN.

Networks created with neighbourhood connectivity demonstrate a steady rise in storage efficiency with increasing connectivity when training with random data though at no stage does the efficiency exceed that of the networks created with random connectivity at equivalent loadings. The results for networks learning the character data are particularly interesting as it appears that the storage efficiency remains largely the same as the level of connectivity increases. It is perhaps not surprising that little benefit is seen from the higher levels of connectivity given that it was shown earlier (c.f. §7.4.2) that the level of local correlation at neighbourhood sizes of 3 and above is very close to the measured level of global correlation.

8.6.3. Attractor Performance

One of the most common performance indicators is that of the ability of the fundamental memories of a network to act as attractors. The measure used is that of Kanter and Sompolinsky (1987) modified as described in chapter 3. The results from which the summaries have been generated are the mean of five simulation runs. The full tables may be found in appendix D.

8.6.3.1. Results for Random (b=0.5) and Geometric Data

Tables 8.8 and 8.9 show the attractor performance analysis summaries for networks learning random data (bias 0.5) and geometric data. Results for each of the levels of connectivity described in table 8.1 are shown. The networks are compared in two ways: Firstly, the effect that the type of training data has on attractor performance is examined (table 8.8). Secondly, the effect of the connectivity strategy is examined with respect to the two types of training data (table 8.9).

The type of data or pattern of connectivity providing the highest attractor performance is given in each case.

Comparing between random data (b=0.5) and geometric data

MCPN	Random connectivity	Neighbourhood connectivity
7.41	Failed to train	Failed to train
21.09	No non-trivial attractor performance using either data type	No non-trivial attractor performance using either data type
39.96	Similar at low loadings. Decreases more quickly for geometric data	No non-trivial attractor performance using either data type
63	Similar at low loadings. Decreases more quickly for geometric data	Higher for random data
89.25	Similar at low loadings. Decreases more quickly for geometric data	Higher for random data

Table 8.8: Results of attractor performance comparisons between random (b=0.5) and geometric data at each of five levels of random or neighbourhood connectivity. The type of data resulting in the highest attractor performance is given for each case.

Table 8.8 summarises, for networks with random or neighbourhood connectivity, the results of comparing the attractor performance of those networks when learning random (b=0.5) data and geometric data.

Networks with random connectivity, where they succeed to train and produce results suitable for study, have attractor performance which is the same at low loadings for both types of data being learnt. The level of attractor performance decreases more rapidly with respect to the loading for the networks that were trained using the geometric data.

For neighbourhood connectivity, the attractor performance is higher for random data than for character data at each of the loadings where a non-trivial attractor performance was achieved.

These results are in line with expectations. Random connectivity should, to a certain degree, nullify any effect that local correlation in the data might have and cause each neuron to see input patterns biased at 0.5 regardless of the data type. It was seen in the analysis of the training data (c.f. chapter 7) however, that the site analysis of the geometric training data revealed that a number of bits had a level of bias throughout the training set above that for the pattern set as a whole. This could account for the quicker fall in attractor performance for the geometric data on random connectivity.

The results for neighbourhood connectivity are unsurprising for a similar reason. Where results exist, the attractor performance is higher for random data than for geometric data. The input patterns seen by each neuron are far more similar to each other when using neighbourhood connectivity with geometric data as when random connectivity is used. Random connectivity allows each neuron to see input patterns that are biased at approximately the level of the whole dataset. The less biased patterns permit the neurons, as perceptrons, to have greater generalisation ability resulting in greater overall attractor performance.

Comparing between random connectivity and neighbourhood connectivity

MCPN	Random data (b=0.5)	Geometric data
7.41	Failed to train	Failed to train
21.09	No non-trivial attractor performance using either connectivity strategy	No non-trivial attractor performance using either connectivity strategy
39.96	Higher for random connectivity	Higher for random connectivity
63	Higher for random connectivity	Higher for random connectivity
89.25	Higher for random connectivity	Higher for random connectivity

Table 8.9: Results of attractor performance comparisons between random and neighbourhood connectivity strategies at each of five levels of connectivity for networks learning random data (b=0.5) and geometric data.. The pattern of connectivity resulting in the highest attractor performance is given for each case.

Table 8.9 again shows results in line with expectations. When learning random data, random connectivity appears to consistently provide better attractor performance in the cases where the results are suitable for analysis. This is easily explained by considering the mean number of connections per neuron. For random connectivity, the actual number of connections for each neuron will be very close to the mean number. For neighbourhood connectivity the constraint that connectivity may not wrap-around at the edges of the network causes corner an edge neurons to have connectivity that is, at times, far below the mean level. It is very likely that it is this reduced connectivity that results in poorer attractor performance for networks connected using a neighbourhood strategy.

The explanation for geometric data is the same as that given above for random data. In addition to the reduced edge and corner connectivity though, we now have the extra problem of highly biased input patterns in the case of neighbourhood connectivity, as was described in the explanation of the results for table 8.8.

8.6.3.2. Results for Random (b=0.8) and Character Data

Tables 8.10 and 8.11 show the attractor performance analysis summaries for networks learning random data (bias 0.8) and character data. Results for each of the levels of connectivity described in table 8.1 are shown. The networks are compared in the same way as those learning random data (bias 0.5) and geometric data.

The type of data or pattern of connectivity providing the highest attractor performance is given in each case.

Comparing between random data (b=0.8) and character data

MCPN	Random connectivity	Neighbourhood connectivity
7.41	Failed to train	Failed to train
21.09	No non-trivial attractor performance	No non-trivial attractor performance
39.96	Higher for random data	Higher for character data
63	Higher for random data	Higher for character data
89.25	Higher for random data	Similar for both data types

Table 8.10: Results of attractor performance comparisons between random (b=0.8) and geometric data at each of five levels of random or neighbourhood connectivity. The type of data resulting in the highest attractor performance is given for each case.

The results shown in table 8.10 are broadly in line with expectations. Networks exhibit better attractor performance when random data is coupled with random connectivity. These results are similar to those shown for unbiased random data trained on networks with random connectivity in that those networks showed similar attractor performance at low loadings but the performance declined faster with increasing pattern load for geometric data. In this case, where biased random patterns and character data have been used, the benefit of random data manifests itself in higher attractor performance values.

It can be seen in the results for networks with neighbourhood connectivity that local connectivity is providing attractor performance benefits when the network is being trained on the locally correlated character data. This is in contrast to the results with unbiased random patterns and geometric data.

Comparing between random connectivity and neighbourhood connectivity

MCPN	Random data (b=0.8)	Character data
7.41	Failed to train	Failed to train
21.09	No non-trivial attractor performance using either connectivity strategy	No non-trivial attractor performance using either connectivity strategy
39.96	Higher for random connectivity	Higher for neighbourhood connectivity
63	Higher for random connectivity	Higher for random connectivity
89.25	Higher for random connectivity	Higher for random connectivity

Table 8.11: Results of attractor performance comparisons between random and neighbourhood connectivity strategies at each of five levels of connectivity for networks learning random data (b=0.8) and character data.. The pattern of connectivity resulting in the highest attractor performance is given for each case.

Table 8.11 again shows that random data is best paired with random connectivity. For all levels of connectivity where a result exists and is non-trivial it can be seen that random connectivity produces higher attractor performance in conjunction with random connectivity.

When learning character data, the effect of the pattern of connectivity is less conclusive. Neighbourhood connectivity results in higher attractor performance in only one case, 39.96 MCPN. Interestingly, this corresponds to the last neighbourhood size at which additional local correlation was seen, according to the training data analysis results presented in §7.4.2.3. Beyond this level of connectivity, random connectivity again appears to be the most beneficial.

8.6.4. Neuron Failure Count

The rate at which neurons fail to learn their particular input patterns once a network's loading gets too high for training to be successful is of particular interest. As mentioned earlier (c.f. §8.4.2), if the rate at which neurons fail with increasing pattern load can be kept low then it is hoped that a small number of additional connections might be sufficient to compensate for the failure and allow a network to stabilise all patterns.

As with previous measures, all values are the mean of 5 simulation runs. The results are presented graphically for this measure so as to better illustrate the rate of increase in neuron failure.

8.6.4.1. Results for Random (b=0.5) and Geometric Data

The following 2 graphs show the rate at which neurons fail to successfully learn their input patterns when attempting to learn random data of bias 0.5. The first graph shows this rate for networks in which random connectivity has been established. The second graph shows the same for networks in which the connectivity forms a local neighbourhood. Each graph shows results for the 5 different levels of connectivity being used.

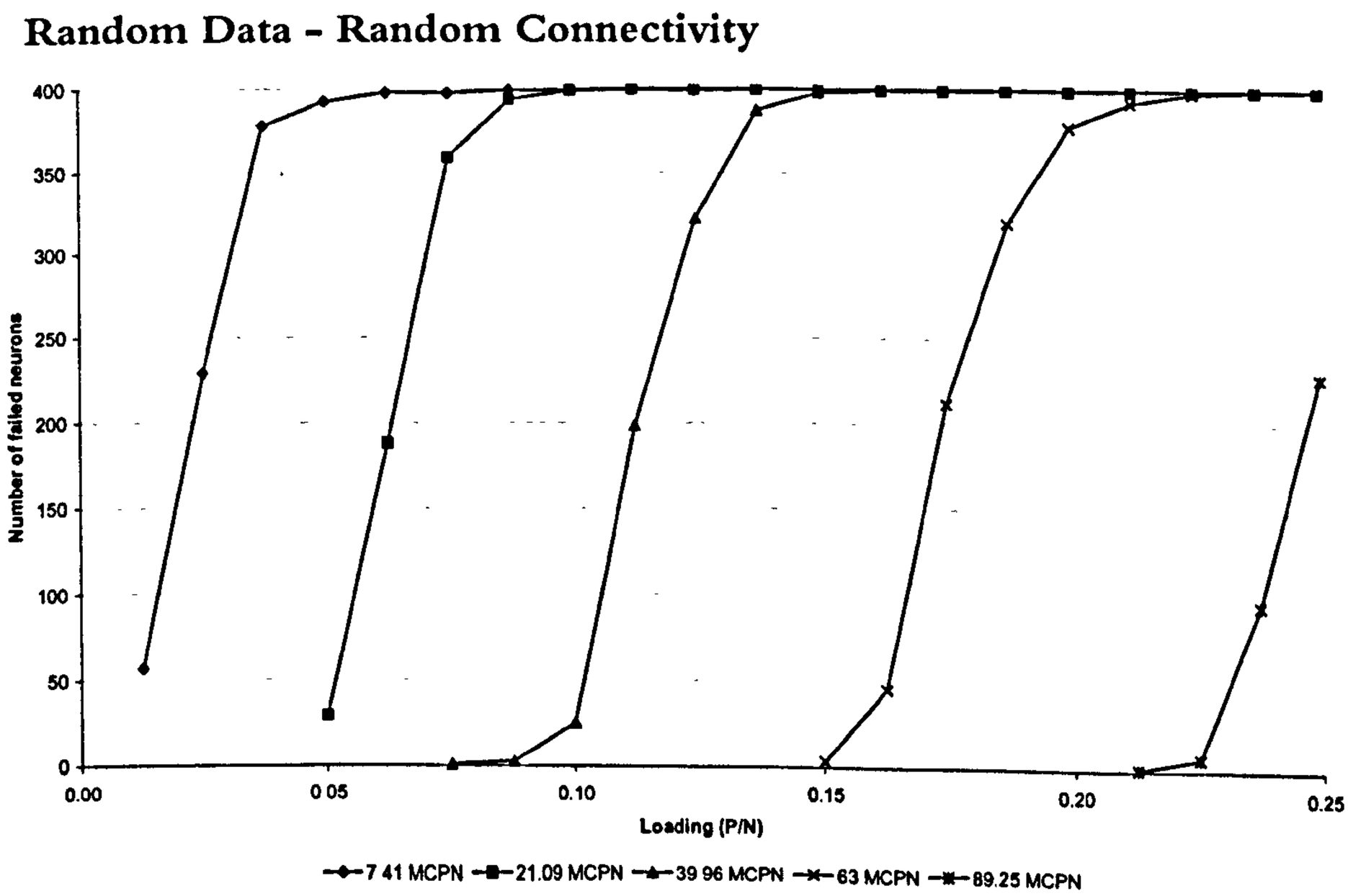


Figure 8.4: Failed neuron count against increasing pattern load for networks constructed with random connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using random (b=0.5) data.

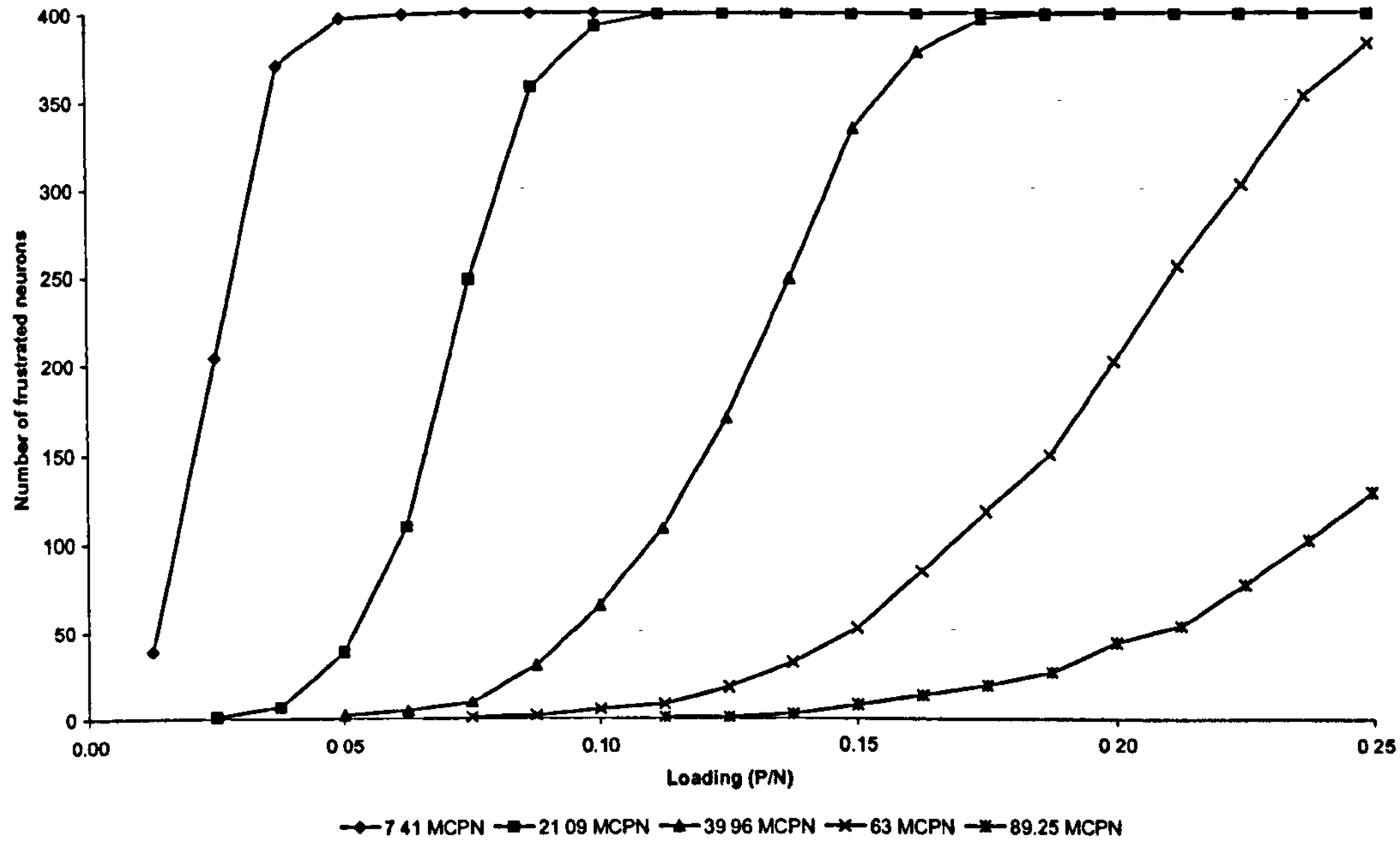


Figure 8.5: Failed neuron count against increasing pattern load for networks constructed with neighbourhood connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using random (b=0.5) data.

The immediate observation from the graphs shown in figures 8.4 and 8.5 (above) is that there is not a great deal of difference in the rate at which neurons fail with increasing pattern load for networks learning random (b=0.5) data. This is the case regardless of whether or not the networks have been constructed with random or neighbourhood connectivity. This is entirely as might be expected. The random nature of the data causes the localised structured connectivity to have little or no effect on the rate of neuron failure.

Examining the graphs more closely does reveal a slight difference between the networks. At low levels of connectivity (7.41 and 21.09 MCPN), the failure rates are near identical. As the level of connectivity increases, the rate of failure becomes gentler. This can be seen in the difference between the graphs when comparing the lines representing 39.96, 63, and 89.25 MCPN. In general, for the networks constructed with neighbourhood connectivity, the point of first failure occurs at a lower loading than for randomly connected networks with the same level of connectivity. An example of this can be clearly seen by examining, in each graph, the line representing a level of connectivity of 39.96 MCPN. For randomly connected networks the point of first failure occurs around a loading of 0.0750 whereas for networks with neighbourhood connectivity failure begins at a loading of approximately

0.0500. This early initial failure is likely to be due to the reduced levels of connectivity of corner and edge neurons.

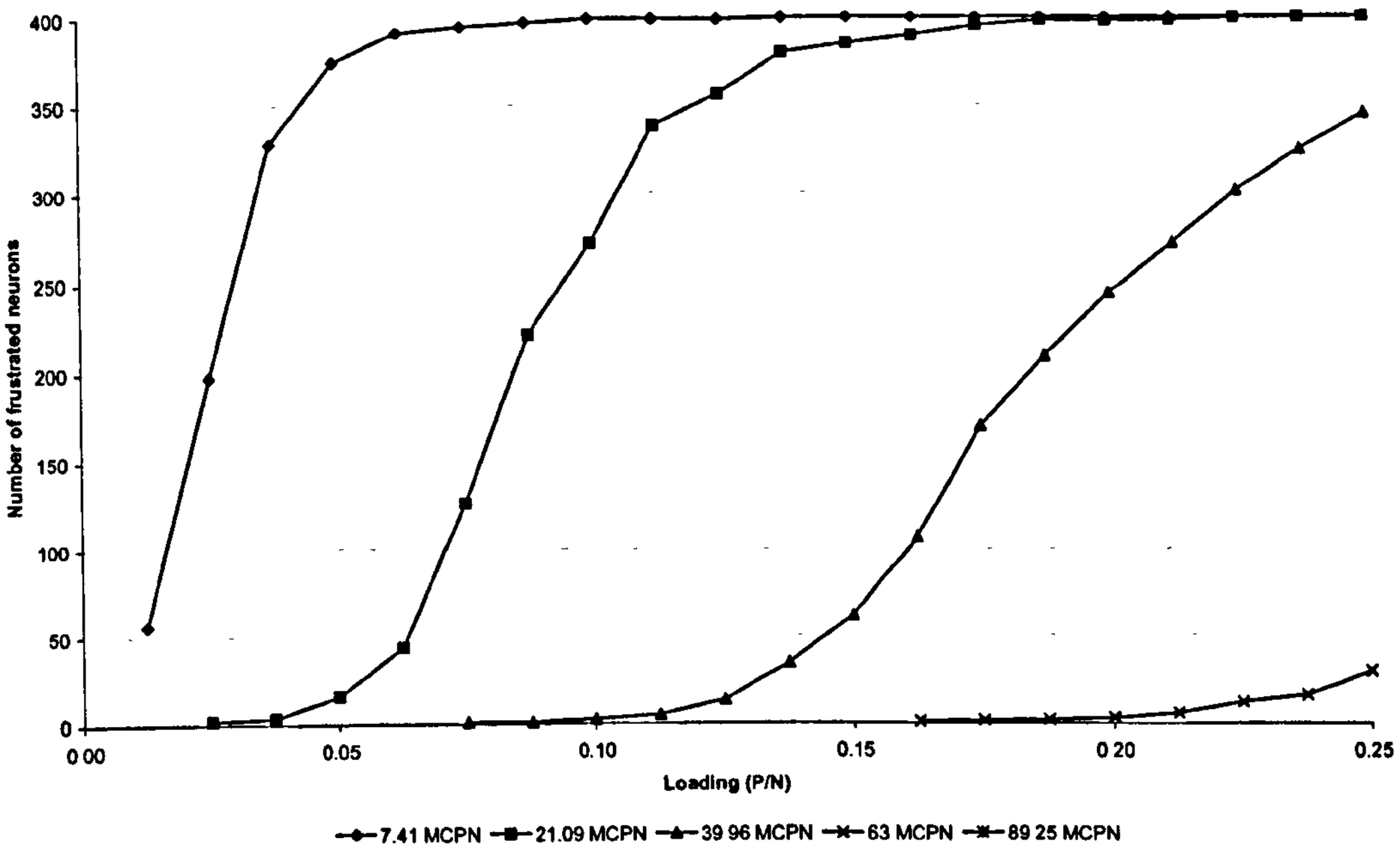


Figure 8.6: Failed neuron count against increasing pattern load for networks constructed with random connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using geometric data.

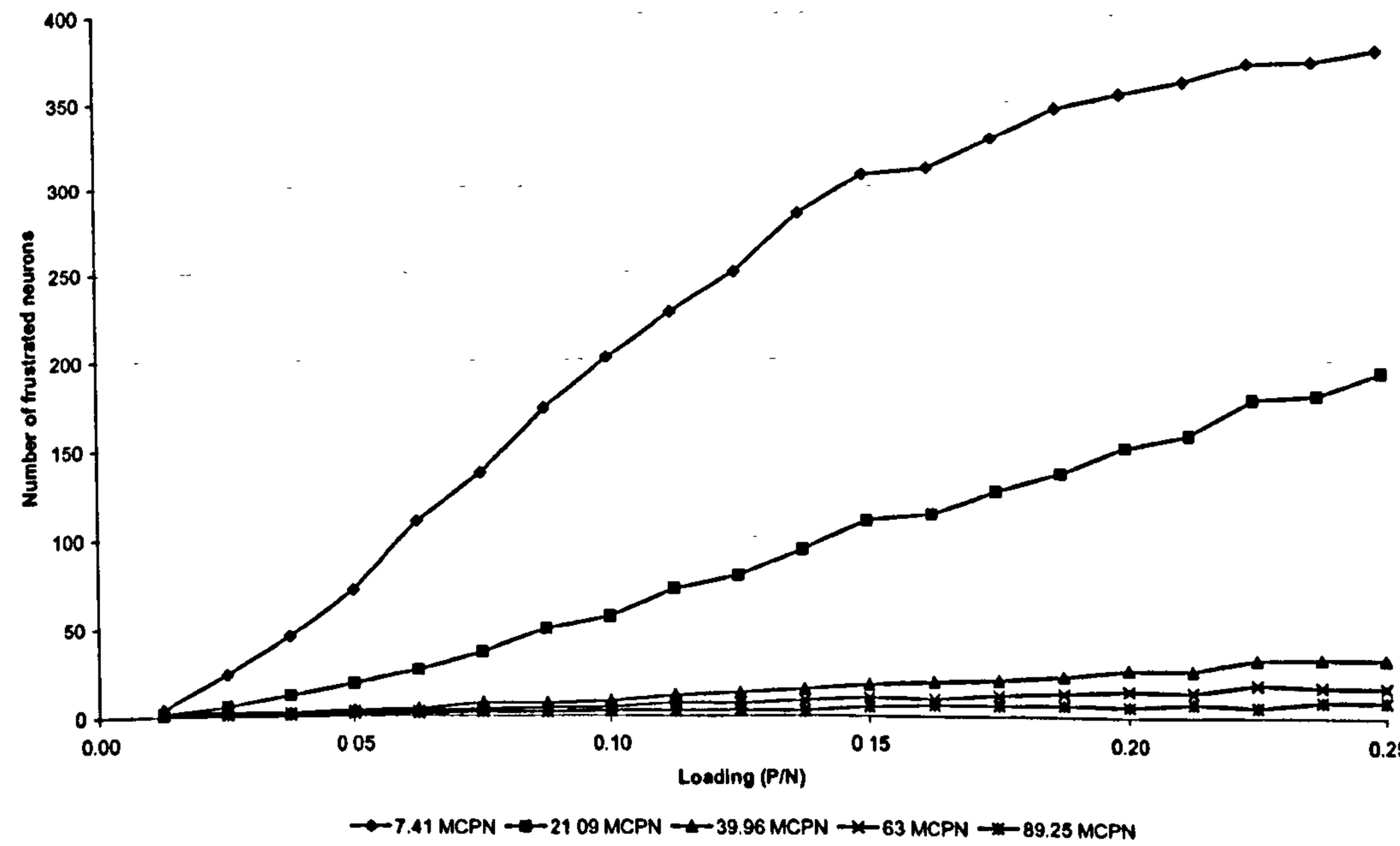


Figure 8.7: Failed neuron count against increasing pattern load for networks constructed with neighbourhood connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using geometric data.

There is a marked difference in appearance between the graphs shown in figures 8.6 and 8.7. The graphs respectively show geometric data trained on

networks with various degrees of random connectivity and on networks created with equivalent levels of neighbourhood connectivity.

The networks with random connectivity follow a similar pattern to that seen for random data on both random and neighbourhood connectivity. Once a loading has been reached at which at least 1 neuron has failed to train, the failure count at subsequent loadings rises rapidly. Although the quantity of failed neurons increases rapidly, this increase becomes less dramatic as the connectivity of the network rises. This can be clearly seen by observing the slope of the lines representing level of connectivity of 39.96 and 63 MCPN. The absence of data for networks with 89.25 MCPN indicates that the network learnt the training patterns successfully even at the highest loading of 0.25 (100 patterns).

The networks with neighbourhood connectivity exhibit particularly interesting characteristics. At the lowest level of connectivity (7.41 MCPN) it can be seen that the speed at which the count of failed neurons increases with respect to increasing pattern load is far less than was seen for any of the other training data/connectivity combinations at the same level of connectivity. This remains true for the 2 subsequent levels of connectivity (21.09 and 39.96 MCPN).

The point of first failure consistently occurs at lower loadings for networks with neighbourhood connectivity then it does for randomly connected networks. All the networks with neighbourhood connectivity exhibit some degree of neuron failure at the lowest loading attempted (=0.0125) regardless of the level of connectivity that has been established.

It can be seen that, for levels of neighbourhood connectivity greater than 39.96 MCPN, there is little further advantage to be gained in termed of the rate of increase in neuron failure count.

8.6.4.2. Results for Random (b=0.8) and Character Data

The following 2 graphs show the neuron failure count against increasing pattern load when attempting to learn random data of bias 0.8. The first graph shows this rate for networks in which random connectivity has been established. The second graph shows the same for networks in which the connectivity forms local neighbourhoods. Each graph shows results for the 5 different levels of connectivity being used.

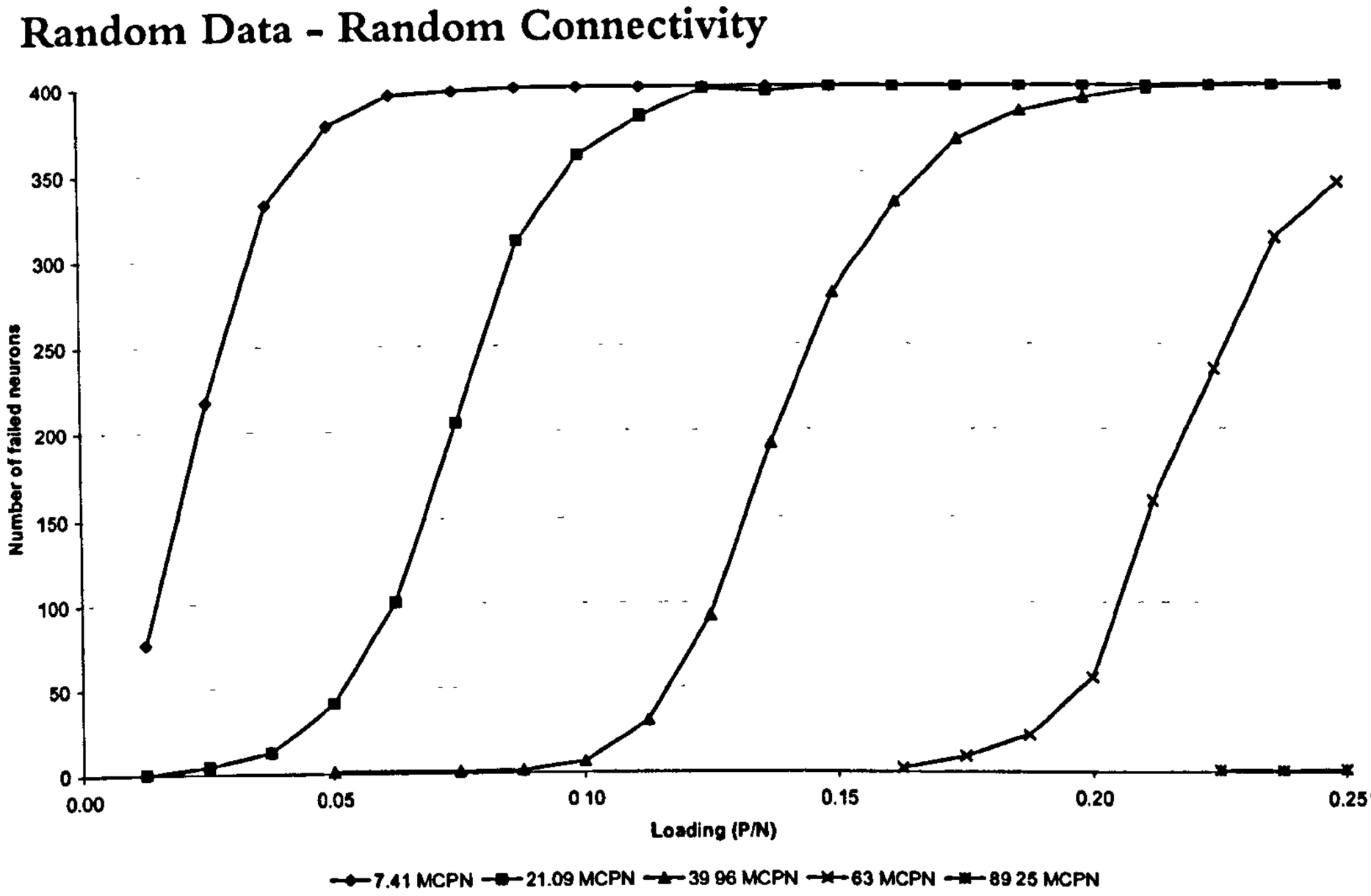


Figure 8.8: Failed neuron count against increasing pattern load for networks constructed with random connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using random (b=0.8) data.

As was the case for networks with unbiased random data there is a strong similarity between the rate of increase in neuron failure count for networks with random and neighbourhood connection topologies at levels of connectivity equal to 7.41, 21.09, and 39.96 MCPN.

At higher levels of connectivity, the networks with neighbourhood connectivity begin to exhibit signs of a lower rate of increase than their randomly connected counterparts. This is evident from the line representing a level of connectivity of 63 MCPN which, in the case of the graph for networks with neighbourhood connectivity, rises more slowly than that for network that are randomly connected.

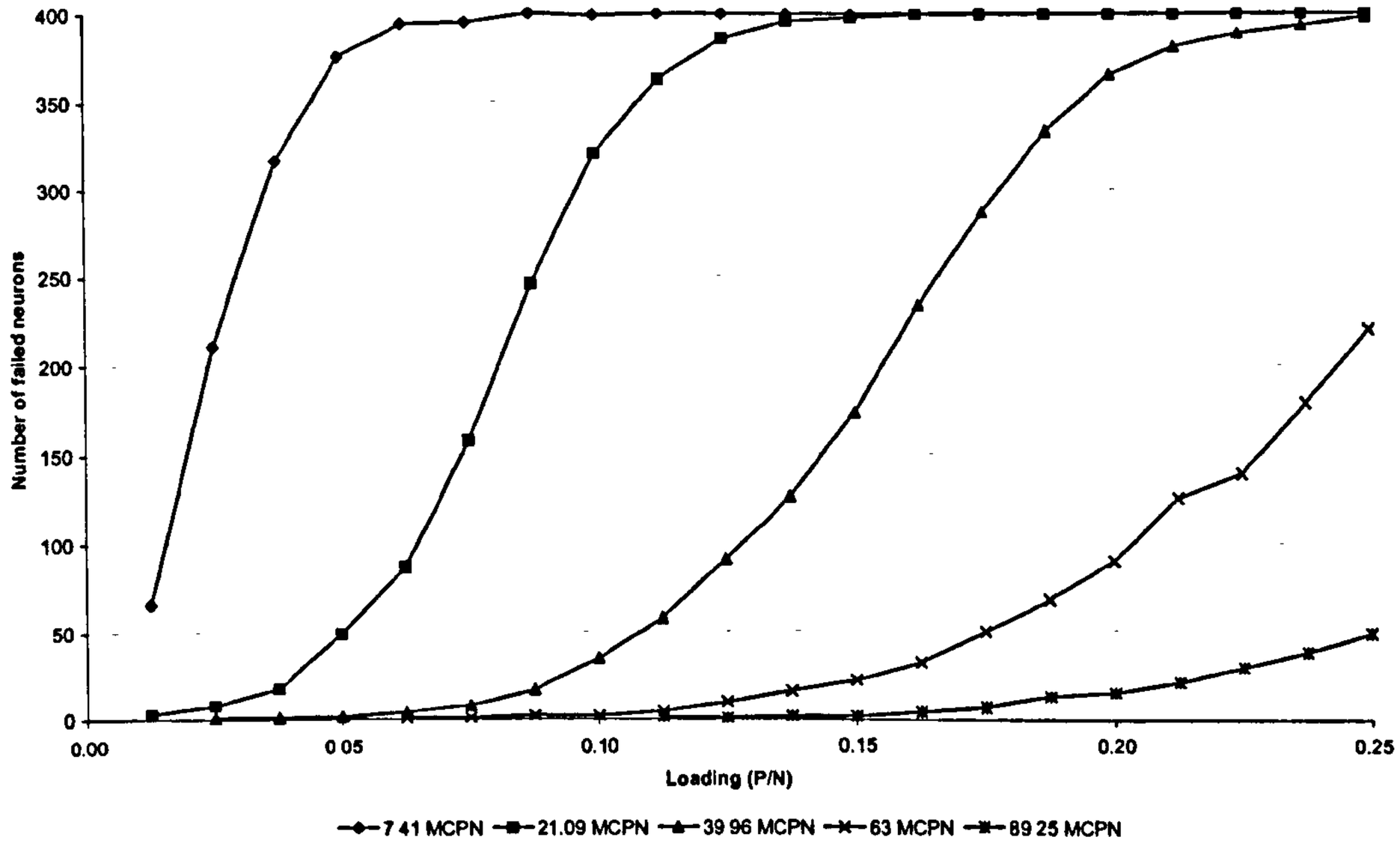


Figure 8.9: Failed neuron count against increasing pattern load for networks constructed with neighbourhood connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using random (b=0.8) data.

Finally, due to the fact that randomly connected networks consistently manage a higher point of first failure, the line representing the highest level of connectivity (89.25 MCPN), is incomplete in the case of figure 8.8. On previous evidence however, it would not be unreasonable to expect a similar rise in neuron failure as was seen at lower levels of connectivity.

The following 2 graphs show the neuron failure count against increasing pattern load when attempting to learn character data. The first graph represents networks with random connectivity whereas the second graph shows results for networks in which the connectivity forms local neighbourhoods.

Character Data - Random Connectivity

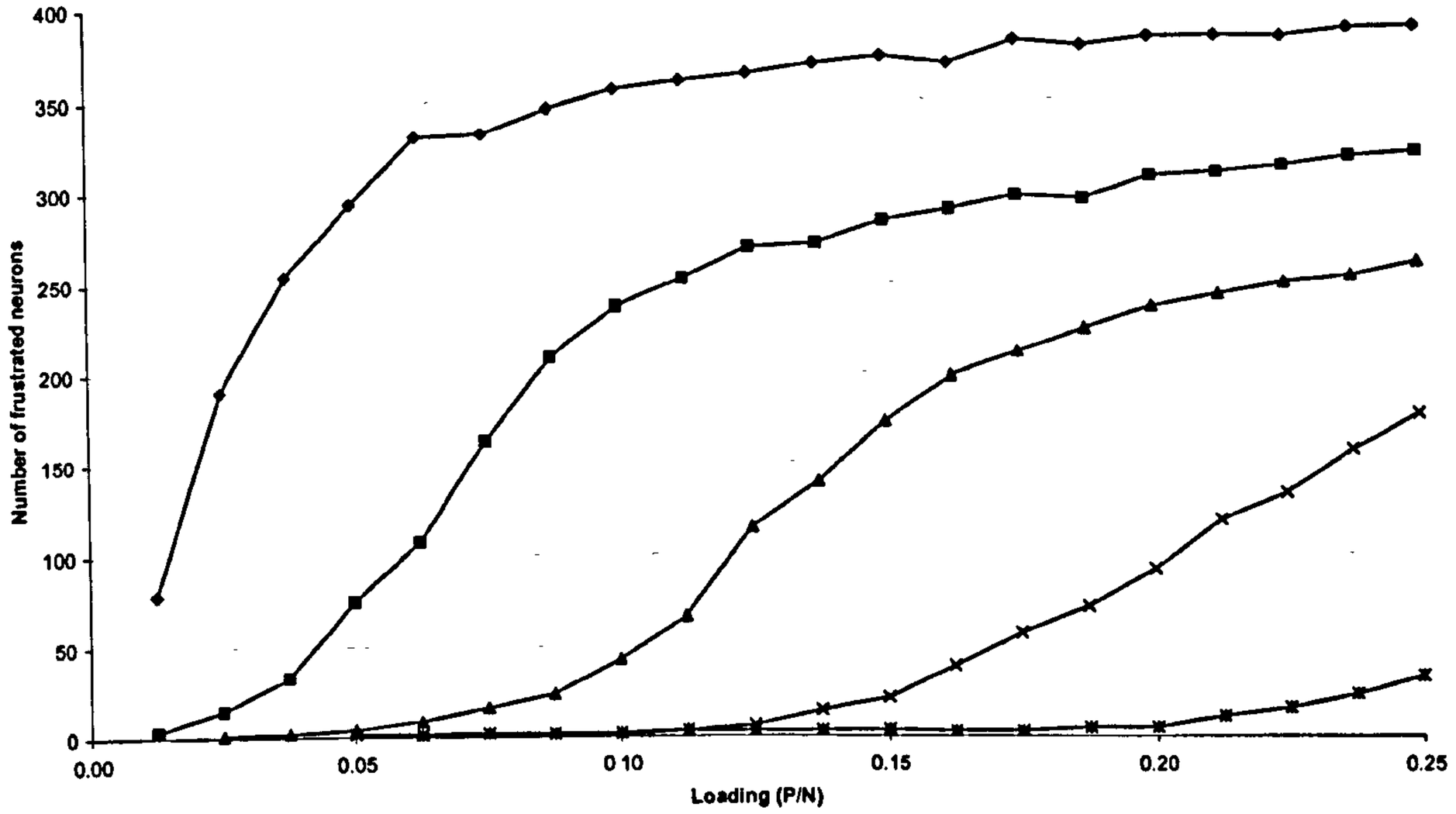


Figure 8.10: Failed neuron count against increasing pattern load for networks constructed with random connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using character data.

There is a clear difference in appearance between figures 8.10 and 8.11. For networks with random connectivity, at the lowest level of connectivity (7.41 MCPN) the rate of increase in the number of failed neurons is initially steep but becomes more shallow beyond a loading of approximately 0.05 (20 patterns). By contrast, the equivalent line for networks with neighbourhood connectivity slopes much more gently and rises roughly linearly with pattern load.

A more stark difference between the 2 types of connectivity can be seen for higher levels of connectivity. The rate of increase in neuron failure count at a level of connectivity of 21.09 MCPN is vastly lower for the locally connected network than for the randomly connected one. A further fall in the rate of increase is seen when moving to neighbourhood connectivity at a level of 39.96 mean connections per neuron. The comparison with the equivalent

level of random connectivity again shows a large difference in favour of neighbourhood connectivity.

Character Data - Neighbourhood Connectivity

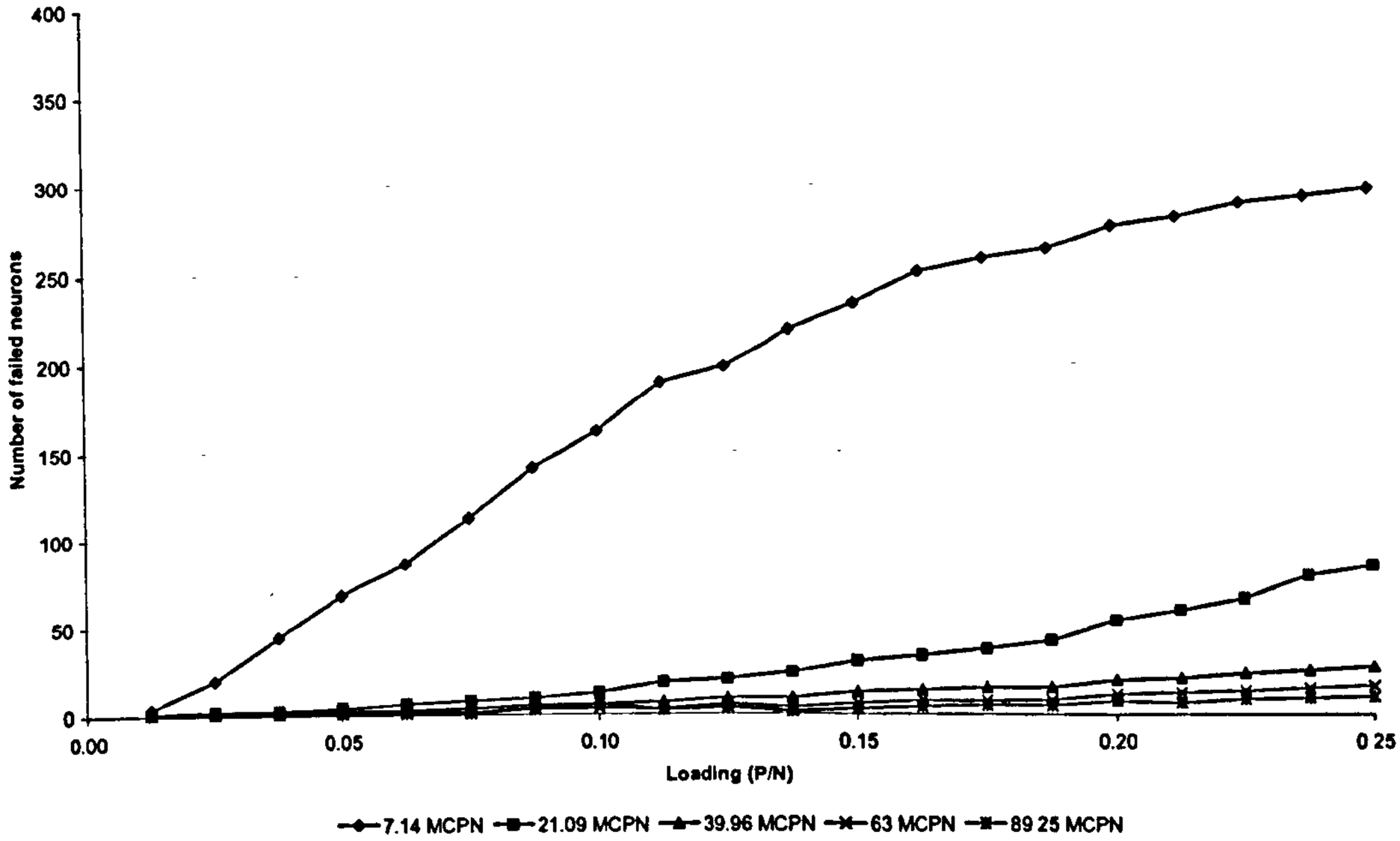


Figure 8.11: Failed neuron count against increasing pattern load for networks constructed with neighbourhood connectivity at levels of 7.41, 21.09, 39.96, 63, and 89.25 mean connections per neuron and trained using character data.

The advantage in adding further local connectivity seems to be less when the level is increased to either 63 or 89.25 MCPN. A level of connectivity of 39.96 MCPN using neighbourhood connectivity is enough to better, in terms of neuron failure count, the higher levels of random connectivity.

As was noted earlier for networks learning geometric data, the networks created with neighbourhood connectivity generally have a point of first failure that occurs much sooner than for randomly connected networks. This does not happen so much with networks learning character data. It can be seen from figures 8.10 and 8.11 that the networks have their point of first failure in roughly the same area.

8.7. Summary and Conclusions

This chapter has reported the results of several performance metrics: capacity, training time, storage efficiency, attractor performance, and neuron failure rate. An evaluation was made for each metric as to the way in which the networks' performance was affected by both the type of data being learnt and the pattern of connectivity in the network.

Capacity

It was seen that the capacity of the networks tested was usually higher when learning random data rather than locally correlated data and this was the case regardless of the pattern of connectivity. In the case of neighbourhood connectivity this result was anticipated due to the likely early failure of corner and edge neurons. The fact that it was also the case for random connectivity is unexpected. It might be assumed that the random connectivity would prevent any local correlation within the patterns having any effect on the network. It was observed however, during the analysis of the training data (c.f. §7.4.3), that some bits in the training data had a higher than average level of activity when examined across the whole training set. It is possible that this could account for the difference in capacity between random and locally correlated data with random connectivity.

There are occasional instances of geometric data resulting in higher capacity than unbiased random data. These are at neighbourhood sizes of 3 and above. It is possible that the reduced contribution made to local correlation by bits at such distances (c.f. §7.4.2.3) could be enough to make the resulting input patterns appear quite random to individual neurons. This would have the effect of levelling the playing field somewhat between the geometric and unbiased random data, especially given that both datasets have the same overall level of bias.

It was mentioned earlier (c.f. 8.6) that capacity does not fully represent the storage capability of locally connected networks due to the reduced edge and corner connectivity. Therefore, these results, while of interest, are not of major importance in evaluating the performance of such networks.

Training Time

Combined with the capacity analysis was the measuring of network training time. For the networks learning either geometric or unbiased random data it was seen that training time was always shorter for random data.

When comparing training times between connectivity topologies it is seen that training time is often shorter for random connectivity but is occasional better for neighbourhood connectivity even when training using unbiased random data.

The networks learning character or biased random data usually exhibit shorter training times for random data regardless of the connection topology. Training time was shorter for character data in only one instance of neighbourhood connectivity – 39.96 MCPN (d=3).

Comparing between connectivity topologies for these two types of data it was seen that random connectivity clearly aided the learning of random data. Correspondingly, local neighbourhood connectivity resulted in shorter training times when character data was being learnt.

Storage Efficiency

The results for measuring storage efficiency clearly indicate that the storage efficiency rises with the mean number of connections per neuron for networks with random connectivity. As the connections are evenly distributed throughout the network this is not surprising.

The notional theoretical maximum capacity for networks of this type is equal to twice the input dimensionality of the neurons in the network. This would be equal to a storage efficiency value of 2. As the theoretical maximum capacity is true only for networks of infinite size it makes sense that the storage efficiency of the randomly connected networks used in this work gets closer to the theoretical maximum as the MCPN rises.

For networks with local connectivity the situation is not so clear. The relatively poor storage efficiency can be linked directly to the seemingly poor capacity. The early failure of some neurons is again affecting the performance of locally correlated networks.

Attractor Performance

The attractor performance of the networks investigated was almost universally higher for random data than for locally correlated data of either type. Equally, random connectivity appeared to provide better attractor performance than local neighbourhood connectivity regardless of the type of data being learnt.

It would therefore seem, at first glance, obvious to view the reduced attractor performance of the locally connected networks as a failure. It is crucial to realise however that two topologies being employed cause individual neurons to see very different types of data. In the randomly connected network the patterns should be regularly spaced in their reduced dimensionality state spaces. In the case of the locally connected systems, the patterns are highly correlated and will be close to one another in the state space. This is likely to have a large impact on the attractor basin size as it directly affects the number of bits one can change in any given pattern before ending up closer to some other pattern.

This is a possible explanation for the greater attractor performance of random data on locally connected networks. The local correlation present in the geometric/character data that causes the similarities when applied to locally connected architectures simply is not present in the random data. The local correlation, while helping to improve the capacity of the neurons as per the theory of Lopez and Schroder (1995), is likely to be detrimentally affecting the generalisation performance of each neuron.

Neuron Failure Count

The concept of useful capacity was introduced earlier (c.f. §8.4.2) in order to provide a measure of capacity that took into account the likely early failure of edge and corner neurons. The rate at which the number of neurons which fail to train rises as pattern load increases provides us with this measure.

This result is the critical point to be made from this chapter. The graphs of neuron failure clearly showed that, for random connectivity, once a level of loading had been reach at which at least one neuron failed to train, the number of failed neurons at subsequent loadings rose very quickly. This was true for both random and locally correlated data. Random data also fared badly on local neighbourhood connectivity.

The key result appears when locally correlated data is trained on networks with local connectivity. For small neighbourhoods (d=1 and d=2) the rate of increase in neuron failure is quite high. However, when locally correlated data is trained on networks with local connectivity, a dramatic change in the rate at which neurons fail to train occurs. At a neighbourhood size greater than d=2, a very slow rate of increase in failed neurons is observed. The implications of this are as follows: for networks with neighbourhoods of d=3 and a loading of =0.2500, 100 patterns are being stored by most neurons with only around 40 inputs to each neuron. This loading is in excess of the $2N$ notional maximum capacity even in the small networks examined as part of this work.

In conclusion, the really significant result in this chapter is the difference in useful capacity between networks with random connectivity and those that are locally connected when learning patterns that exhibit both intra- and inter-pattern local correlation. This strongly suggests that it is possible to tailor connectivity to a particular type of data and particularly for image data, which is likely to possess considerable local correlation as a consequence of the spatial and temporal continuity of nature, local connectivity seems especially useful.

The investigation of the way in which networks with small numbers of failed neurons might be compensated for such failure is the subject of the following chapter.

9. INCREASING PERFORMANCE THROUGH INCREASING CONNECTIVITY

9.1. Introduction

The previous chapter successfully demonstrated a particular advantage to using local connectivity in networks to be used to learn patterns possessing local correlation. The results showed that, for networks with local connectivity, the rise in the number of failed neurons present at increasing levels of pattern load was slower than was the case for networks with identical levels of random connectivity although the point of first failure occurred sooner.

Two points need to be made regarding neuron failure:

a) The point of first failure is not as important as might be initially thought because of the relatively small input dimensionality to each neuron introduced by the sparse connectivity. This is particularly evident at the edges and corners of the network. The point of first failure only becomes relevant if its location is consistent over a very large number of training sets.

b) The rate at which the number of neurons that fail at each pattern load increases with respect to the rising pattern load is important because this value is considered over a large number of network sites.

The failed neuron count and pattern load can be used together to develop the concept of *useful capacity* (c.f. §8.4.2). The useful capacity is high when a large number of patterns can be stored with few neuron failures occurring as the failed neurons can likely be corrected at relatively low cost in terms of extra connectivity.

As some neurons do fail to train this chapter is concerned with the investigation of a technique by which it might be possible to compensate for this failure.

The results in the previous chapter showed that the greatest fall in the rate of neuron failure count occurred when using neighbourhood connectivity between networks with d=1 and d=2, and between those with d=2 and d=3, when learning the geometric data. The same was true of character data but the fall was less marked between d=2 and d=3 than it was for the geometric data. Despite the large number of failed neurons at the lower levels of connectivity it remains a possibility that even after correcting the failed neurons the overall level of connectivity might still be extremely low.

9.2. Structure of the Investigation

This investigation is presented in two stages. Stage one involves the stabilisation of training patterns in networks where a number of neurons have failed to train. Stage two examines the way in which attractor performance changes with respect to increasing levels of connectivity beyond that required to simply stabilise the patterns.

The core of the investigation involves a modified version of the Symmetric Local Learning rule used in earlier work. Training occurs in phases: the first phase trains the network using the existing connectivity. The neurons that fail to learn their input patterns are recorded and passed to the second phase.

The second training phase takes the list of failed neurons and adds a new connection between each of them and some other neuron chosen at random to which a connection does not already exist. The symmetry of connections is maintained for the sake of the network dynamics. Having added an extra connection to each of the failed neurons the network undergoes a new training phase with all weights re-initialised.

The two phases are repeated until the training phase reports that all neurons are correctly classifying their input patterns. Upon successful training, the attractor performance is measured using the modified Kanter & Sompolinsky measure described in chapter 4. As the training patterns have only just been stabilised it is expected that the attractor performance will be very low. It is not the attractor performance that is of primary interest however; the key measure is the quantity of extra connectivity required in order to stabilise the training patterns.

The second stage of the investigation covered by this chapter looks at the effect of further, additional connectivity on attractor performance. Having established a pattern of connectivity suitable for stabilising the training patterns, additional random connectivity is added. At regular intervals, the network is retrained and the attractor performance measured. As the initial level of local connectivity can be established using relatively few connections it is hoped that, at similar levels of connectivity, the networks where connections have initially been created locally and where further connectivity has then been added will outperform those where the connectivity is totally random.

9.3. Stabilising Training Patterns in Networks with Failed Neurons

In the previous chapter it was shown that training networks with sparse local connectivity and a high loading of correlated data resulted in a much lower failed neuron count when compared with networks with random connectivity or those being trained with random patterns. The question to be answered by the first stage of the investigation presented in this chapter is how much extra connectivity is required to permit the neurons that failed previously to now correctly classify the set of input patterns presented to them?

The networks used were again 400 neurons in size, arranged on a 20-by-20 grid. The training patterns used are the geometric and character data introduced in chapter 8. Connectivity is initially established as a square neighbourhood around each neuron at distances ranging from 1 to 5.

Several results are presented:

a) The mean number of connections per neuron (MCPN) after stabilisation. This result indicates how much random connectivity was required to be added to the network in order to allow the failed neurons to classify their input patterns correctly.

b) The post-stabilisation storage efficiency of the network. This result shows the storage efficiency of the network taking into account the extra connectivity that was added to correct the failed neurons.

c) The post-stabilisation attractor performance of the network. Using the modified Kanter and Sompolinsky measure described in chapter 4, this result shows the attractor performance of the network once the failed neurons have been corrected.

d) The number of training phases required to stabilise the training patterns. As the correction of failed neurons takes place in phases as described in the previous section, the number of these phases that were necessary is reported.

The values used at each of the pattern loading and neighbourhood sizes is the mean of five simulation runs using a different set of training data in each instance.

9.3.1. Geometric Data

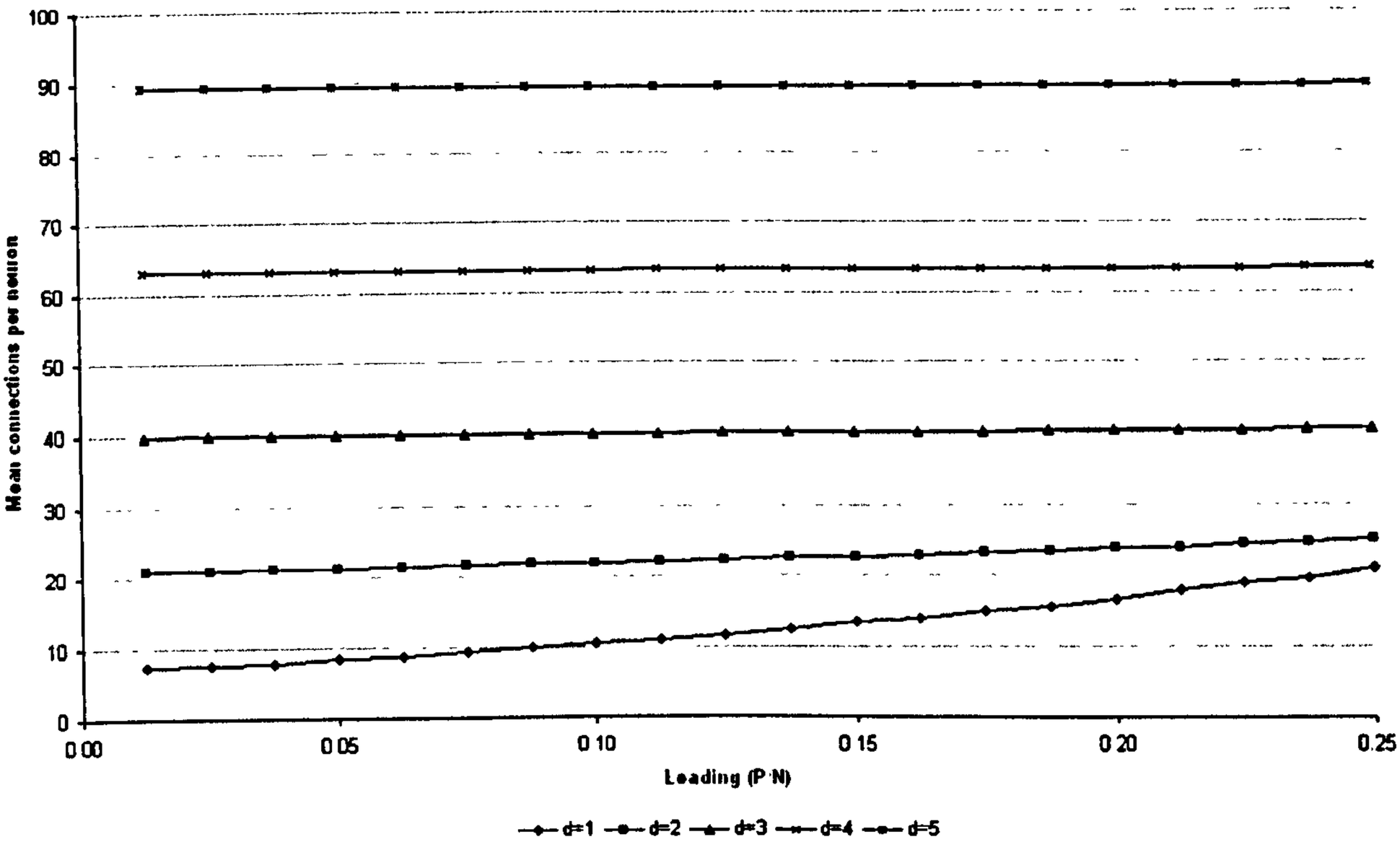


Figure 9.1: Mean number of connections per neuron after stabilisation of failed neurons for geometric data pattern loads of 0.0125 to 0.2500 learnt by networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.1 shows the mean number of connections per neuron at the point at which all failed neurons have been stabilised. Geometric data pattern loads of 0.0125 to 0.2500 (5 to 100 patterns) were used on networks in which neighbourhood connectivity was initially established at sizes from 1 to 5.

The graph shows that only networks with neighbourhoods of size d=1 and d=2 required more than a trivial amount of extra connectivity to stabilise the failed neurons. At d=1, the networks required more than a doubling in connectivity by the time the loading had reached 0.2500 (100 patterns). At d=2, the increase required is not so great being only approximately 25%.

The amount of connectivity required to stabilise 100 patterns in the d=1 networks is approximately equal to the level of pre-stabilisation connectivity in networks with neighbourhoods at d=2.

It is apparent, therefore, that the networks with the lowest level of initial connectivity, despite requiring a large amount of extra connectivity in order to stabilise failed neurons, require less overall connectivity to make their training sets stable. This occurs because the extra connectivity is targeted specifically at those neurons that have failed and this will result in a non-uniform distribution of connections throughout the network.

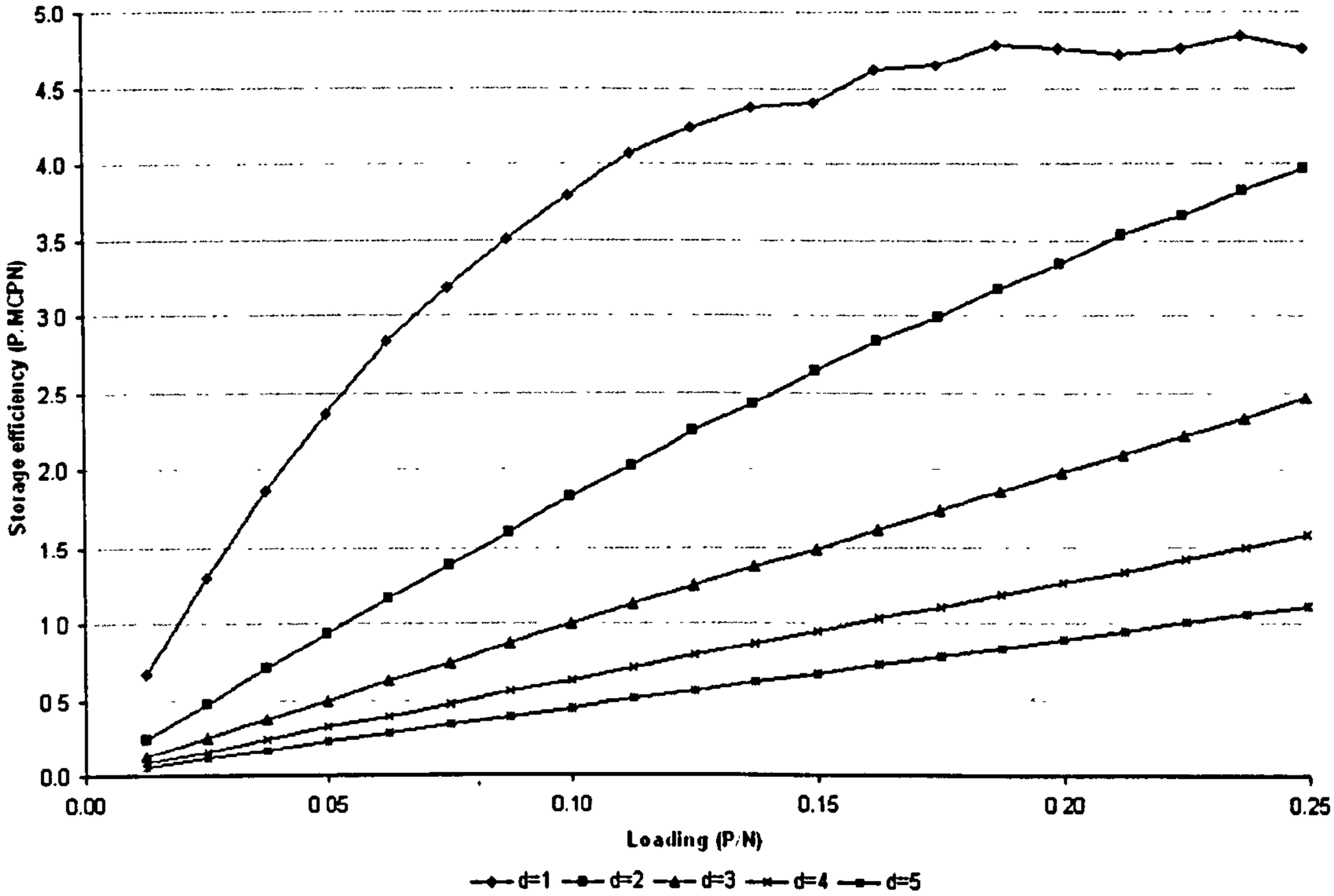


Figure 9.2: Post-stabilisation storage efficiency for geometric data pattern loads of 0.0125 to 0.2500 using networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.2 shows the storage efficiency of each network calculated as the ratio of the number of successfully stored patterns to the mean number of connections per neuron. Each value represents a network created with a level of initial neighbourhood connectivity ranging from d=1 to d=5 and trained using a pattern load of between 0.0125 and 0.2500 (5 and 100 patterns).

The most efficient networks at any of the tested pattern loads were those created with neighbourhood connectivity at d=1. Networks with subsequent neighbourhood sizes become progressively less efficient as the level of connectivity increases.

It is interesting to note that, despite the large amount of extra connectivity required for the networks with the smallest neighbourhoods, the final level of connectivity permits these networks to be the most efficient.

In every case, the efficiency rises as the pattern load increases. The rise in the efficiency of networks with neighbourhood connectivity at d=1 however, appears to slow significantly at around a loading of 0.1125 (45 patterns) and gives the appearance of beginning to level off.

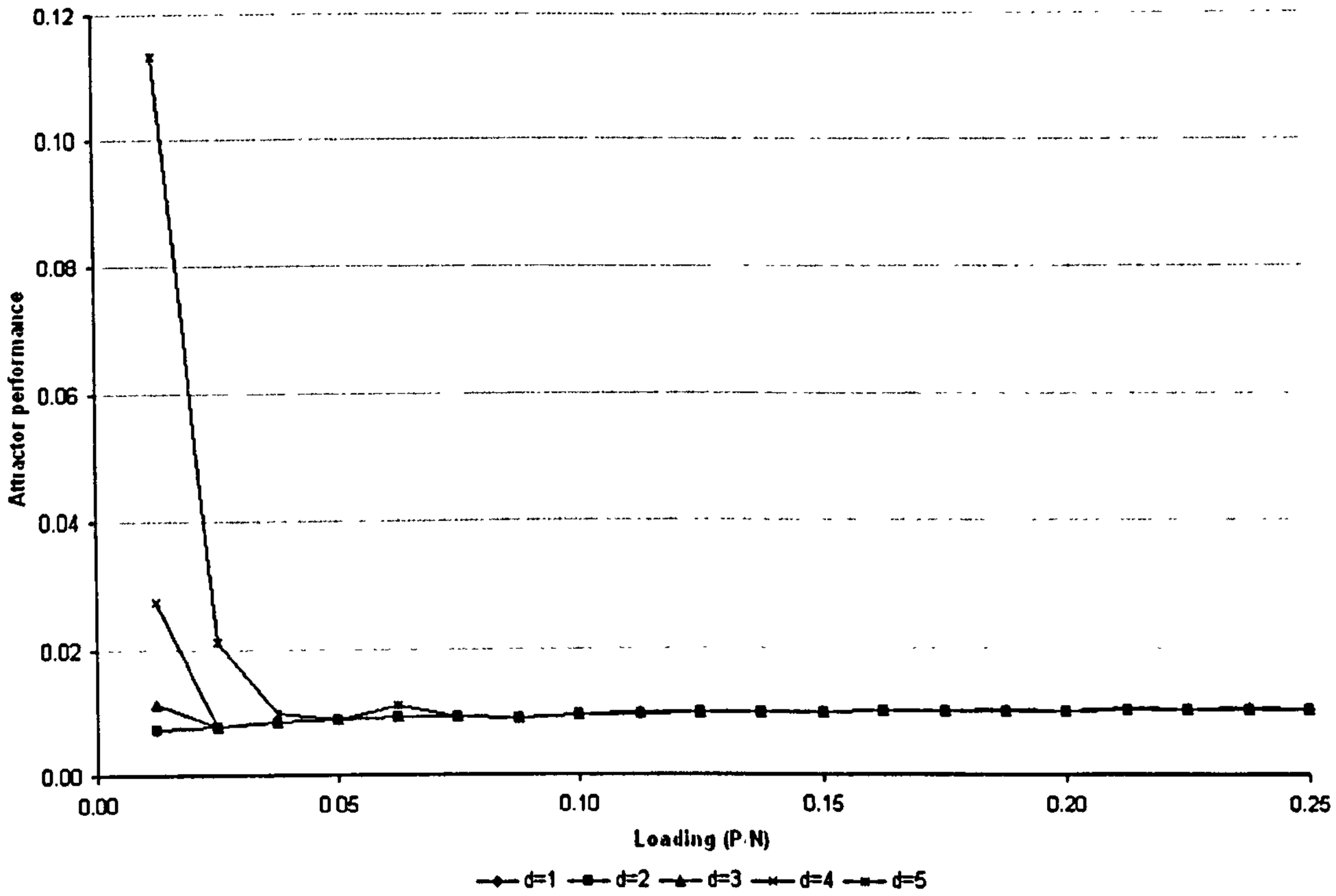


Figure 9.3: Post-stabilisation attractor performance for geometric data pattern loads of 0.0125 to 0.2500 using networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.3 shows the attractor performance of the networks measured after all neurons that initially failed have been corrected. The fact that the attractor performance is at a very similar non-zero level for a significant number of the values obtained could be indicative of either random successes or rounding error in the calculation of the measure. This phenomenon was also seen in the investigation of sparsely connected networks (c.f. chapter 8) where the attractor performance measured also fell to a non-zero value below which it would fall no further. The ability of a network to correct a single corrupt bit in a single stored pattern is sufficient to produce a non-zero value using the Kanter and Sompolinsky (1987) measure. As each value in the figure 9.3 is itself the mean of five simulation runs and the connectivity is somewhat tailored to the training data it is unlikely that a true zero value will occur at the pattern loading levels being used.

Attractor performance above the minimum level exists only for the highest neighbourhood sizes of d=4 and d=5, and the lowest pattern loads of 0.0125 and 0.0250 (5 and 10 patterns). As the networks have only just reached a point where all the training patterns are stable, poor attractor performance is unsurprising.

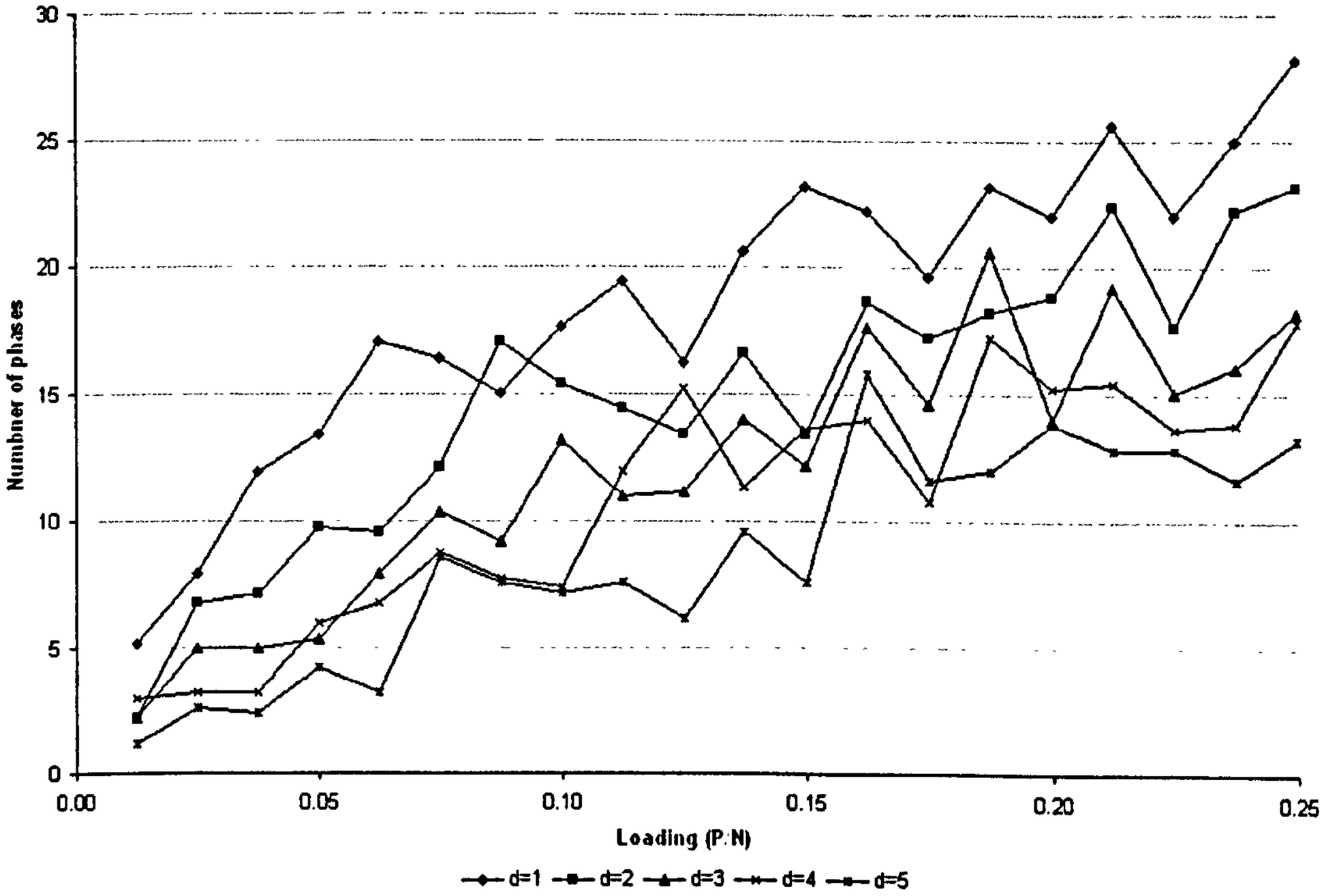


Figure 9.4: Mean number of training phases for geometric data pattern loads of 0.0125 to 0.2500 using networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.4 shows that whilst it is evident that there is a great deal of variability in the results despite the averaging; plotting the number of training phases results in an increase in the number of training phases required as the pattern loading increases.

The lowest number of phases occurs when the pattern load is low and the initial neighbourhood connectivity is high (d=5). The greatest number of phases is required at the opposite end of the scale where the pattern load is at its highest and the neighbourhood size is low (d=1).

The number of training phases required to stabilise the training patterns is dependant therefore, on both the pattern load and the distance at which the initial neighbourhood connectivity is established.

9.3.2. Character Data

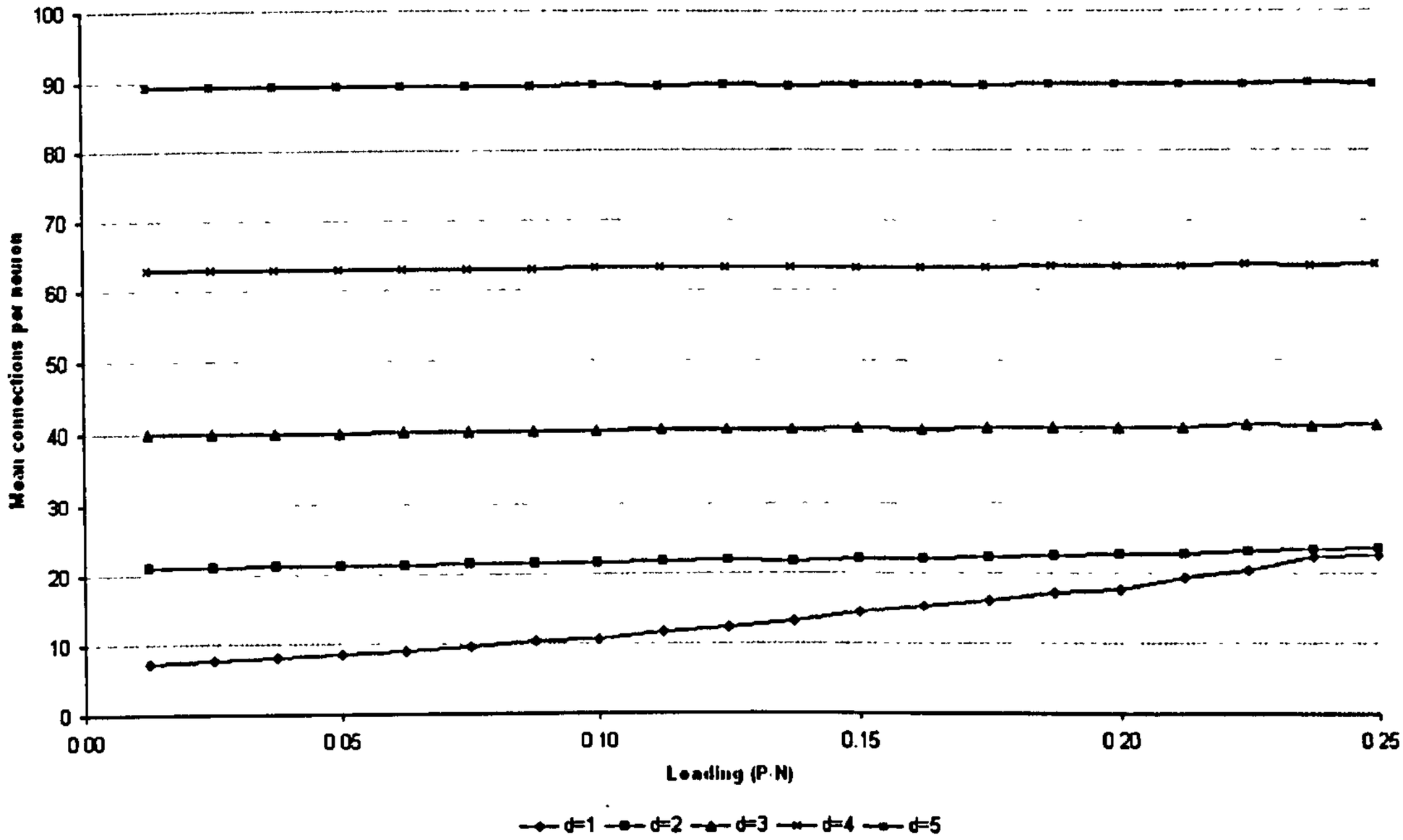


Figure 9.5: Mean number of connections per neuron after stabilisation of failed neurons for character data pattern loads of 0.0125 to 0.2500 learnt by networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.5 shows the mean number of connections per neuron at the point at which all failed neurons have been stabilised. Character data pattern loads of 0.0125 to 0.2500 were used on networks in which neighbourhood connectivity was initially established at sizes from 1 to 5.

The results for character data are very similar to those shown earlier (c.f. figure 9.1) for geometric data.

Only networks with initial neighbourhood connectivity at d=1 and d=2 require a significant amount of extra connectivity in order to stabilise the failed neurons. Again, the post-compensation level of connectivity for d=1 at α=0.2500 (100 patterns) is very close to the pre-compensation connectivity for networks with d=2.

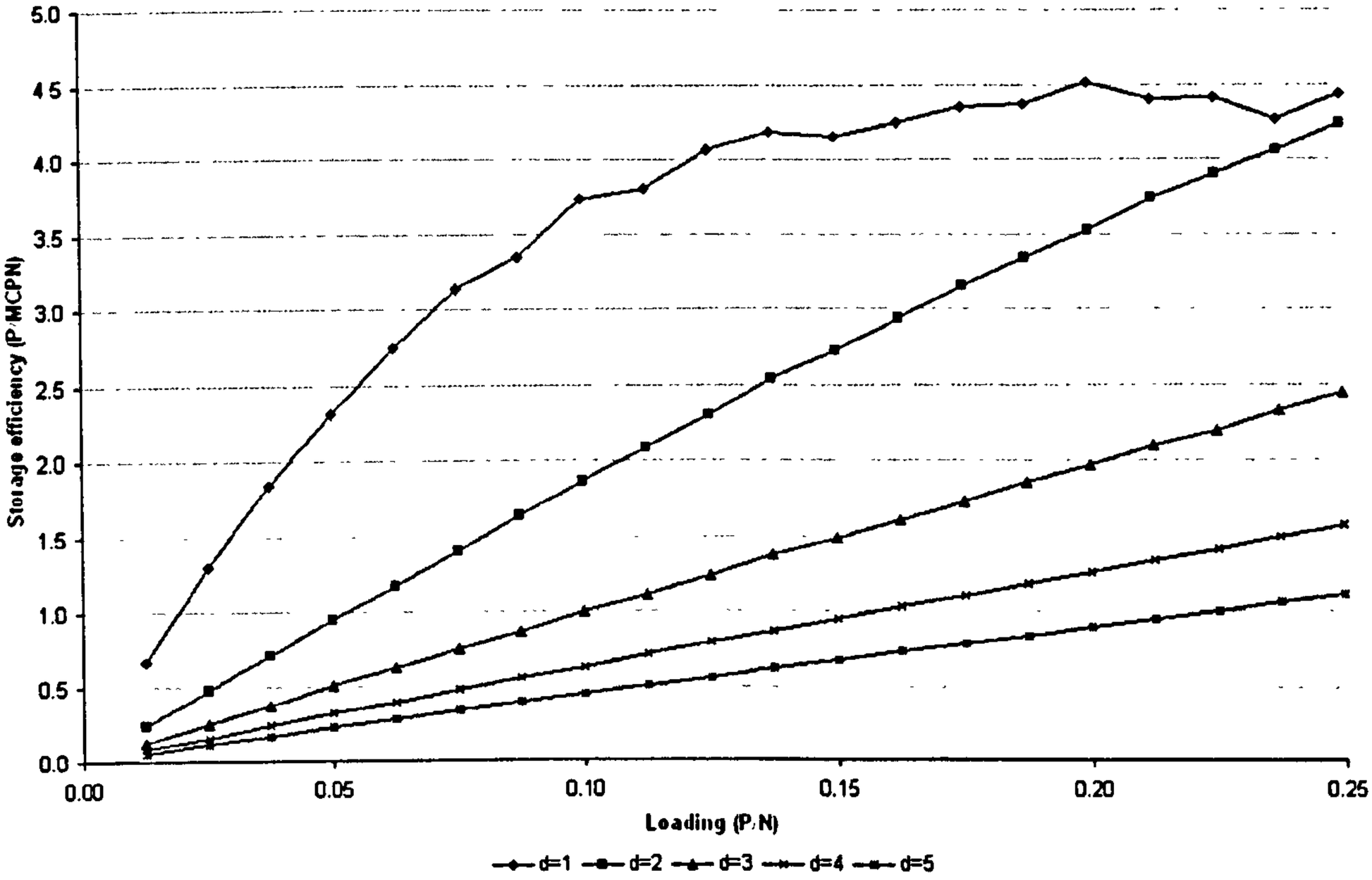


Figure 9.6: Post-stabilisation storage efficiency for character data pattern loads of 0.0125 to 0.2500 using networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.6 shows the storage efficiency of each network calculated as the ratio of the number of successfully stored patterns to the mean number of connections per neuron. Each value represents a network created with a level of initial neighbourhood connectivity ranging from d=1 to d=5 and trained using a pattern load of between 0.0125 (5 patterns) and 0.2500 (100 patterns).

The most storage efficient network at any of the tested levels of pattern loading were those in which the initial level of neighbourhood connectivity was at a minimum (d=1). As was seen to be the case when training with pattern sets based on the geometric data, the overall final level of connectivity in these networks was below that of networks with larger initial neighbourhoods despite the large proportion of extra connectivity required in order to stabilise the training set.

As the level of local neighbourhood connectivity increases, the requirement for extra connectivity falls (c.f. figure 9.5) but the total level of connectivity is such that the storage efficiency is always less than for those with neighbourhoods of d=1.

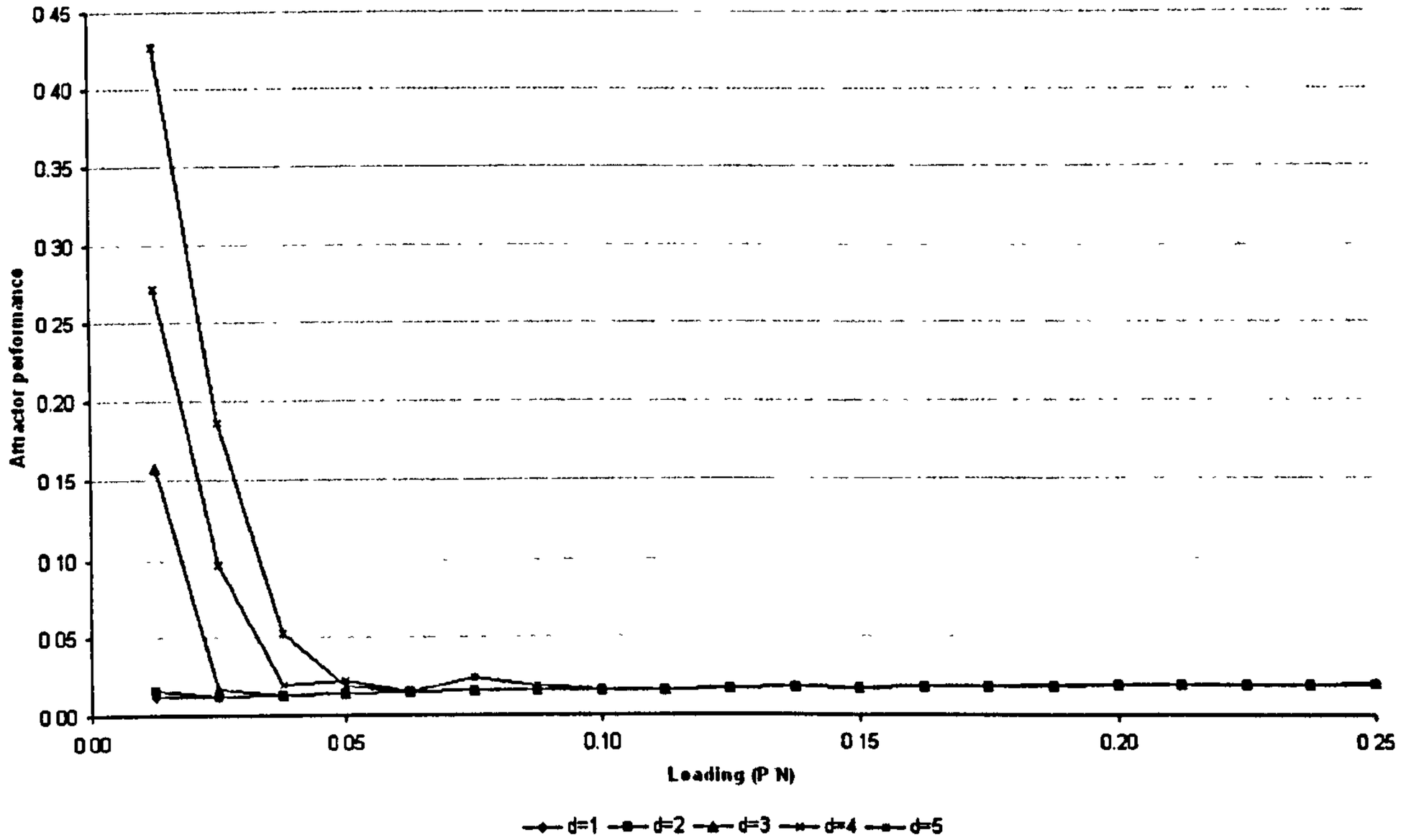


Figure 9.7: Post-stabilisation attractor performance for character data pattern loads of 0.0125 to 0.2500 using networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.7 shows the attractor performance of the networks measured after all neurons that initially failed have been corrected. The levelling out of the attractor performance to a consistent non-zero value was seen earlier for the geometric data and can be attributed to the same causes.

The results for geometric data showed attractor performance above this minimum level for only very low loadings and large neighbourhood sizes and similar behaviour is evident for character data. Attractor performance above the minimum level only occurs for networks with initial neighbourhood connectivity at sizes d=3, 4, and 5 and all easily discernable attractor performance has vanished by a loading of 0.0500 (20 patterns) in the best case of d=5.

No attractor performance of any significance is present for networks with initial neighbourhood connectivity at distances d=1 and d=2. This was also the case for networks learning geometric data.

Where the attractor performance is non-negligible it is consistently higher for the character data than for the geometric data at equivalent pattern loads and neighbourhood size.

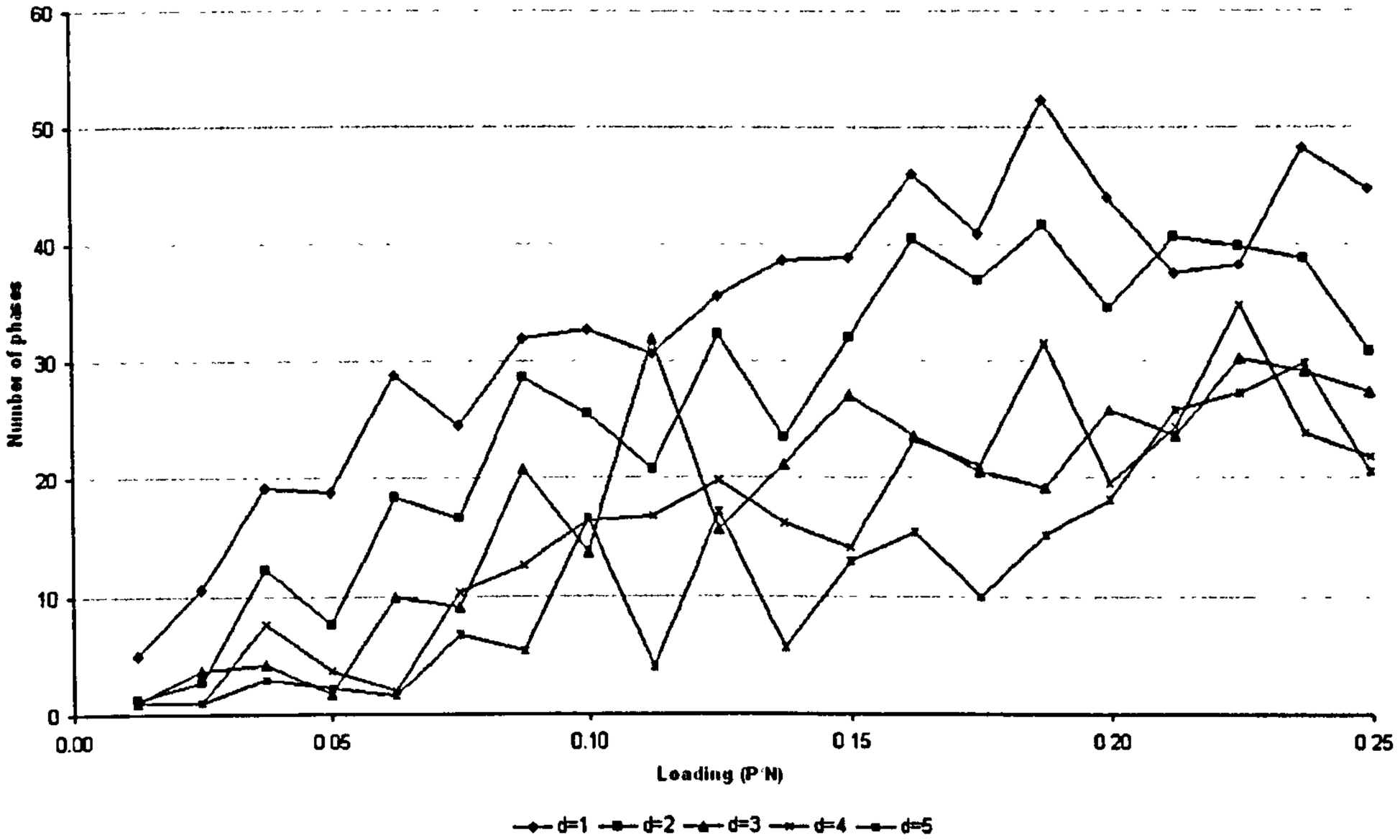


Figure 9.8: Mean training phase count for character data pattern loads of 0.0125 to 0.2500 using networks with initial neighbourhood connectivity established at distances 1 to 5.

Figure 9.8 shows the mean training phase count for five simulation runs at each combination of pattern load and neighbourhood size.

In examining figure 9.8 it should be borne in mind that the phase count range has doubled from that which was used for the geometric data earlier (c.f. figure 9.4). Using geometric data the number of phases required ranged from 0 to 30. For character data this range increases and is now between 0 and 60.

The feature of figure 9.8 that is of most interest is the similarity it bears to figure 9.4 which represents the same information for geometric training data. Given that the number of phases axis represents twice the range as in figure 9.4 the implication is that at equivalent loadings and neighbourhood sizes the number of training phases required is approximately double that required for geometric data.

9.4. Improving Attractor Performance with Further Connectivity

The previous set of experiments was designed to investigate the amount of extra connectivity required to stabilise any neurons that might have failed when connectivity was initially established following a strict neighbourhood strategy.

The second stage of this investigation examined how the attractor performance improved when, after having added connectivity into the network to compensate for the failed neurons, further connectivity was introduced.

It was hoped that by initially establishing the connectivity using the neighbourhood strategy, an improvement in attractor performance would be seen over those networks where the same level of connectivity was established purely randomly.

The networks used were 400 neurons in size, arranged as if on a 20-by-20 grid. The training patterns used were derived from the geometric and character data introduced in chapter 8. Connectivity was initially established as a square neighbourhood around each neuron at distances ranging from 1 to 5. The networks were then trained using the Symmetric Local Learning rule and any failed neurons compensated with extra connectivity as per the previous experiment.

Once a stable network was obtained, more random connectivity was gradually added to it. At particular levels of connectivity, the network was retrained and the attractor performance measured at that point. The levels at which attractor performance should be re-measured was set to be 5% connectivity intervals. The new connectivity was added symmetrically to maintain simple update dynamics.

The justification for adding this extra connectivity is as follows. It is likely that, with the exception of neurons that have been compensated with extra connectivity, the neurons will be seeing input patterns which look very similar. It is in fact possible that a neuron, as a perceptron, might be being required to classifying a set of input patterns that fall into only a single class, i.e. all the patterns have the same output. If either of these situation is the case, the generalisation performance of the neurons could be very poor and will result in the network displaying correspondingly poor attractor performance.

It was seen in previous work (c.f. §8.6.3) that random connectivity resulted in either greater attractor performance or that the attractor performance decreased more slowly with rising pattern load than was the case with neighbourhood

connectivity. This further supports the idea that some random connectivity might be essential for reasonable attractor performance.

Graphs of the results of networks trained at loadings of α=0.0125, 0.1250, and 0.2500 are presented. These loadings correspond to 5, 50, and 100 patterns respectively. Values for all loadings tested may be found in appendix E. The values in the tables and those plotted here are taken as the mean of 5 network simulations. Results for networks with neighbourhood sizes d=1 to d=5 are shown. Additionally, the results for networks with purely random connectivity are shown for comparison, these have been denoted d=0.

9.4.1. Geometric Data

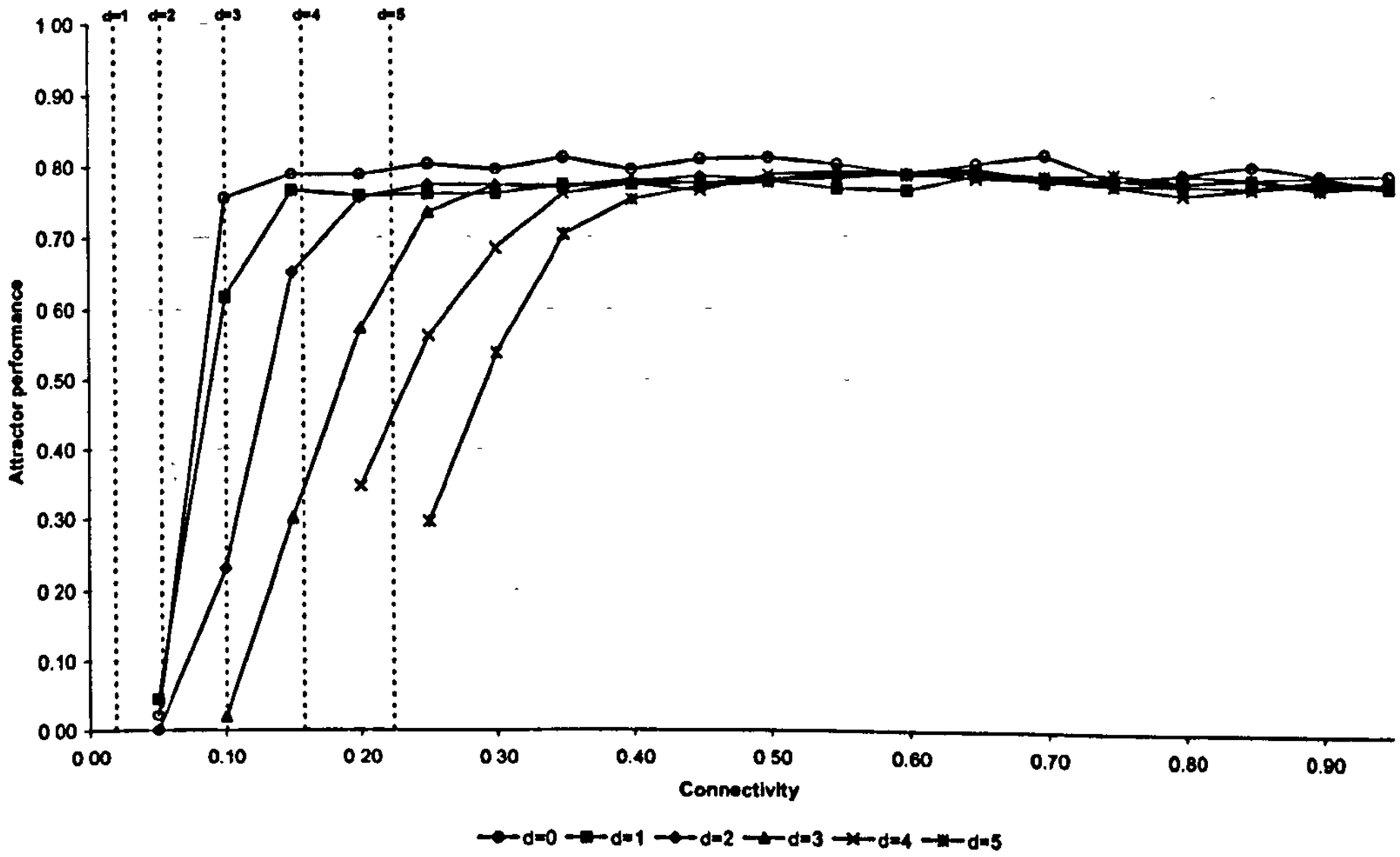


Figure 9.9: The attractor performance of networks initially connected using the local neighbourhood strategy and with further symmetric connectivity added at random. Training patterns were from the geometric data set. Attractor performance is shown at 5% connectivity intervals. The dashed vertical lines represent the level of neighbourhood connectivity before either compensatory or additional random connectivity was added. The loading on the network is α=0.0125 (5 patterns).

Figure 9.9 shows the effect of additional random symmetric connectivity on the attractor performance of networks in which connectivity was initially established using a local neighbourhood strategy. The loading on the network is α=0.0125 (5 patterns).

The clearest point to be made from figure 9.9 is that none of the networks, regardless of the level of initial local connectivity, were able achieve a level of attractor performance in excess of $R\approx0.80$. It is very likely that with this level of

loading, this is the maximum attainable attractor performance. The lowest level of connectivity at which this level of attractor performance was reached was for networks in which the connectivity was established wholly at random, d=0. The level of connectivity required was 0.15.

Networks with the smallest neighbourhoods (d=1) achieve almost the same level of attractor performance at the same loading. The level of connectivity initially established using the local neighbourhood strategy is only 0.01 however. The quantity of extra random connectivity added in order to obtain near-maximum attractor performance will have dwarfed that which is part of the local neighbourhoods. It is highly probable that this network is almost indistinguishable from it's totally randomly connected counterpart in terms of the pattern of connections.

The requirement for large amounts of random connectivity in order to achieve high attractor performance exists for networks of all the neighbourhood sizes examined.

An interesting, though easily explained, point should be made regarding networks with initial neighbourhoods of d=4 and d=5. The proximity of the corresponding dashed vertical lines, representing the initial connectivity levels, to the first plotted value for those networks would seem to indicate that little extra random connectivity was required in order to obtain some non-trivial attractor performance. This is somewhat misleading; the local neighbourhoods for these networks are larger than the range at which significant local correlations exist in the training data (c.f. chapter 7). This means the individual neurons may see input patterns that incorporate some non-correlated portions of the training data. This is not dissimilar to having a network with smaller local neighbourhoods in which some random compensation has taken place to correct failed neurons. In short, the networks with neighbourhoods of size d=4 and d=5 already possess a degree of *effectively* random connectivity even though no compensatory connectivity has been required to stabilise the training patterns.

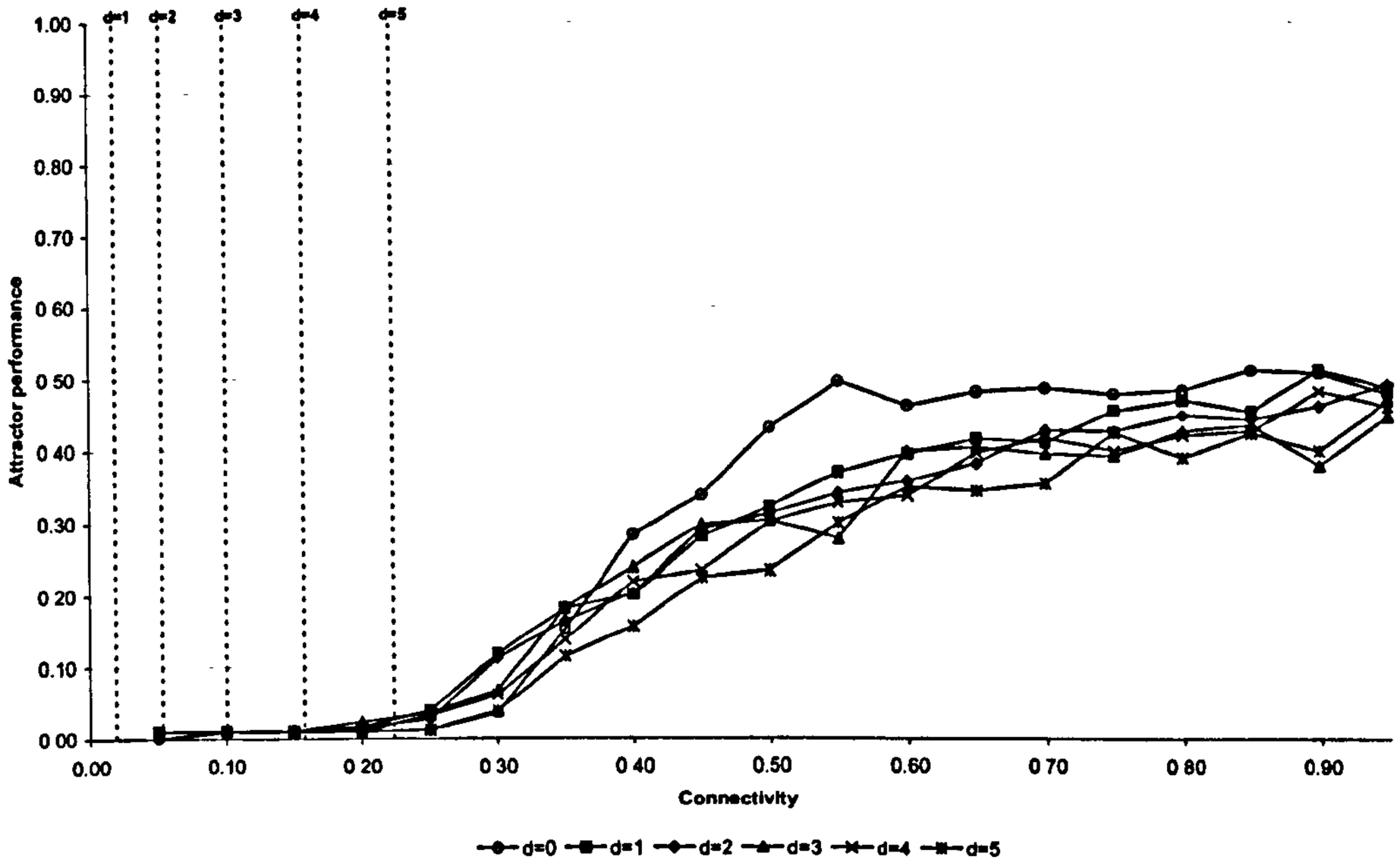


Figure 9.10: The attractor performance of networks initially connected using the local neighbourhood strategy and with further symmetric connectivity added at random. Training patterns were from the geometric data set. Attractor performance is shown at 5% connectivity intervals. The dashed vertical lines represent the level of neighbourhood connectivity before either compensatory or additional random connectivity was added. The loading on the network is =0.1250 (50 patterns).

Figure 9.10 is as 9.9 (previous page) but represents networks with a loading of α=0.1250 (50 patterns).

As different as the plots for networks with a loading of α=0.0125 were, the most obvious feature of figure 9.10 is the similarity shown between networks with some degree of initial neighbourhood connectivity. If the same is true at this loading as at the previous then it can be assumed that the d=0 network indicated the maximum attractor performance attainable. If this is indeed the case, it can be seen that none of the locally connected networks achieve the maximum attractor performance until they are very nearly fully connected. By contrast, the randomly connected network, d=0, reaches a value close to its maximum at a level of connectivity of 0.55.

It would seem that, despite the large amount of random connectivity being introduced to the networks, the effect of the local connectivity may be to fix some of the individual input patterns to the neurons in a small area of the pattern space. These input patterns can therefore not help but be closer together than would be the case if all the input sources are chosen at random. The closer proximity of the input patterns detrimentally affects the generalisation

performance of each neuron and this affects the attractor performance of the network as a whole.

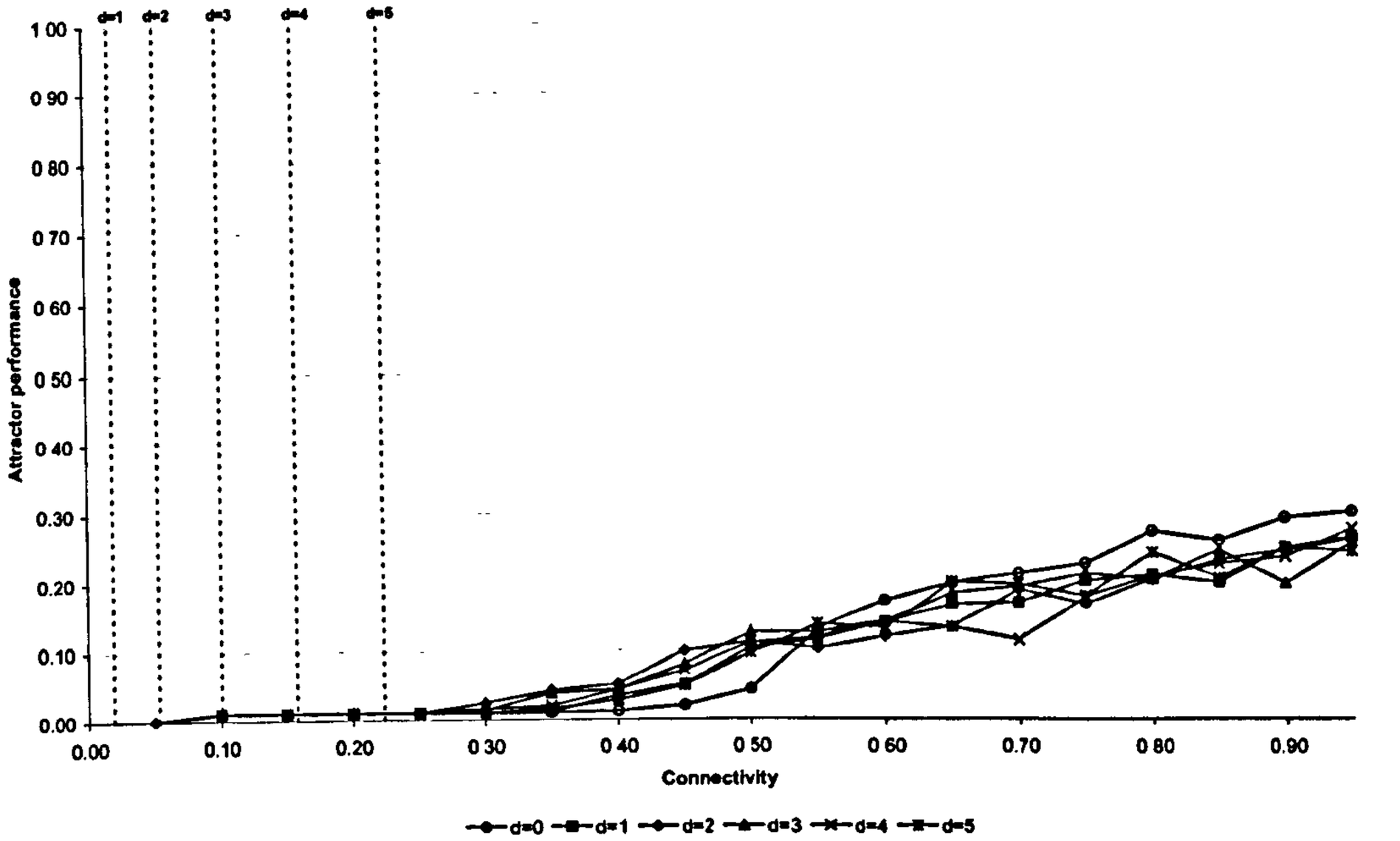


Figure 9.11: The attractor performance of networks initially connected using the local neighbourhood strategy and with further symmetric connectivity added at random. Training patterns were from the geometric data set. Attractor performance is shown at 5% connectivity intervals. The dashed vertical lines represent the level of neighbourhood connectivity before either compensatory or additional random connectivity was added. The loading on the network is •=0.2500 (100 patterns).

Figure 9.11 is as 9.10 (previous page) but represents networks with a loading of α=0.2500 (100 patterns).

It can be seen from the above figure that very little exists to differentiate between the performances of the networks with partial local connectivity. There is, however, a range of connectivity where all of the partially locally connected networks outperform the random connected ones. Between levels of connectivity of 0.35 and 0.55 the line representing randomly connected networks is below those of the locally connected networks. As the level of random connectivity increases, there becomes less to disambiguate between the pattern of connectivity in each of the networks and the performance of the randomly connected networks actually rises above that of the others.

9.4.2. Character Data

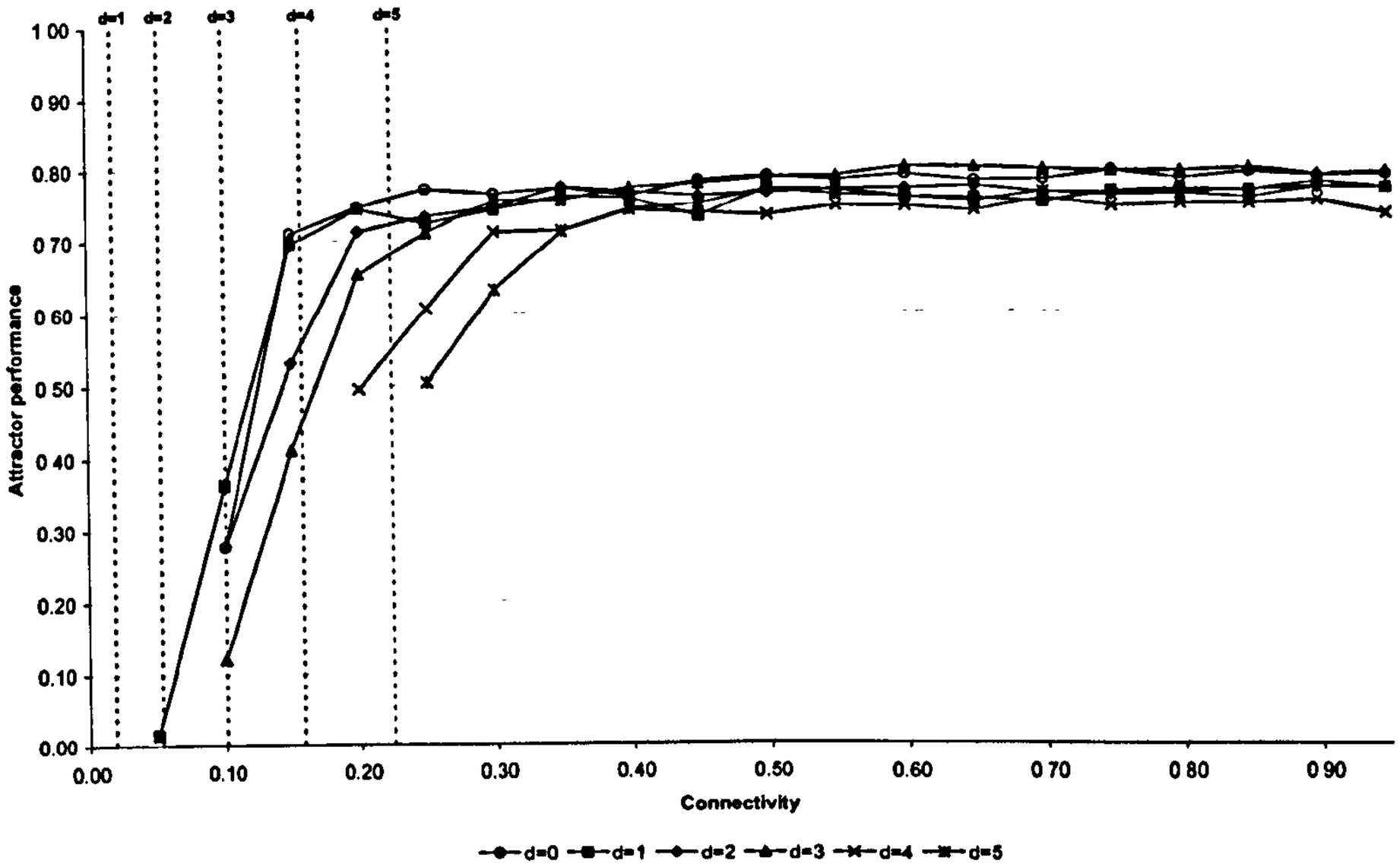


Figure 9.12: The attractor performance of networks initially connected using the local neighbourhood strategy and with further symmetric connectivity added at random. Training patterns were from the character data set. Attractor performance is shown at 5% connectivity intervals. The dashed vertical lines represent the level of neighbourhood connectivity before either compensatory or additional random connectivity was added. The loading on the network is α=0.0125 (5 patterns).

Figure 9.12 shows the effect of additional random symmetric connectivity on the attractor performance of networks in which connectivity was initially established using a local neighbourhood strategy. The loading on the network is α=0.0125 (5 patterns). The training patterns used were those derived from the character data set.

Comparing figure 9.12 with the corresponding graph for geometric data (figure 9.9) it can be seen that the similarity between the lines plotted is much greater than was the case for geometric data. The likely cause of this is the increased level of global correlation in the data compared with that present in the geometric data. The increased global correlation may be mitigating the impact of having a neighbourhood size larger than the range at which local correlation is greatest. The increased global correlation will only have an effect for medium range connectivity however. It was seen when analysing the training data (c.f. chapter 7) that neighbourhood sizes greater than d=3 for character data would begin to introduce local correlation at a level below that of global correlation. This is the probable reason that the lines representing initial neighbourhood connectivity at d=4 and d=5 stand apart from the others.

It is interesting to note that at the lowest level of connectivity for which the attractor performance was measured, the value for the attractor performance is greater than that of the networks learning geometric data. It is likely that this is due to the overall level of bias of the training data which, once the level of random connectivity is sufficiently high, will be close to that of the reduced dimensionality input patterns to each neuron. It is known from Gardner (1988) that patterns with higher bias should result in greater attractor performance. This was shown to be the case in the experimental results of Davey and Hunt (2000).

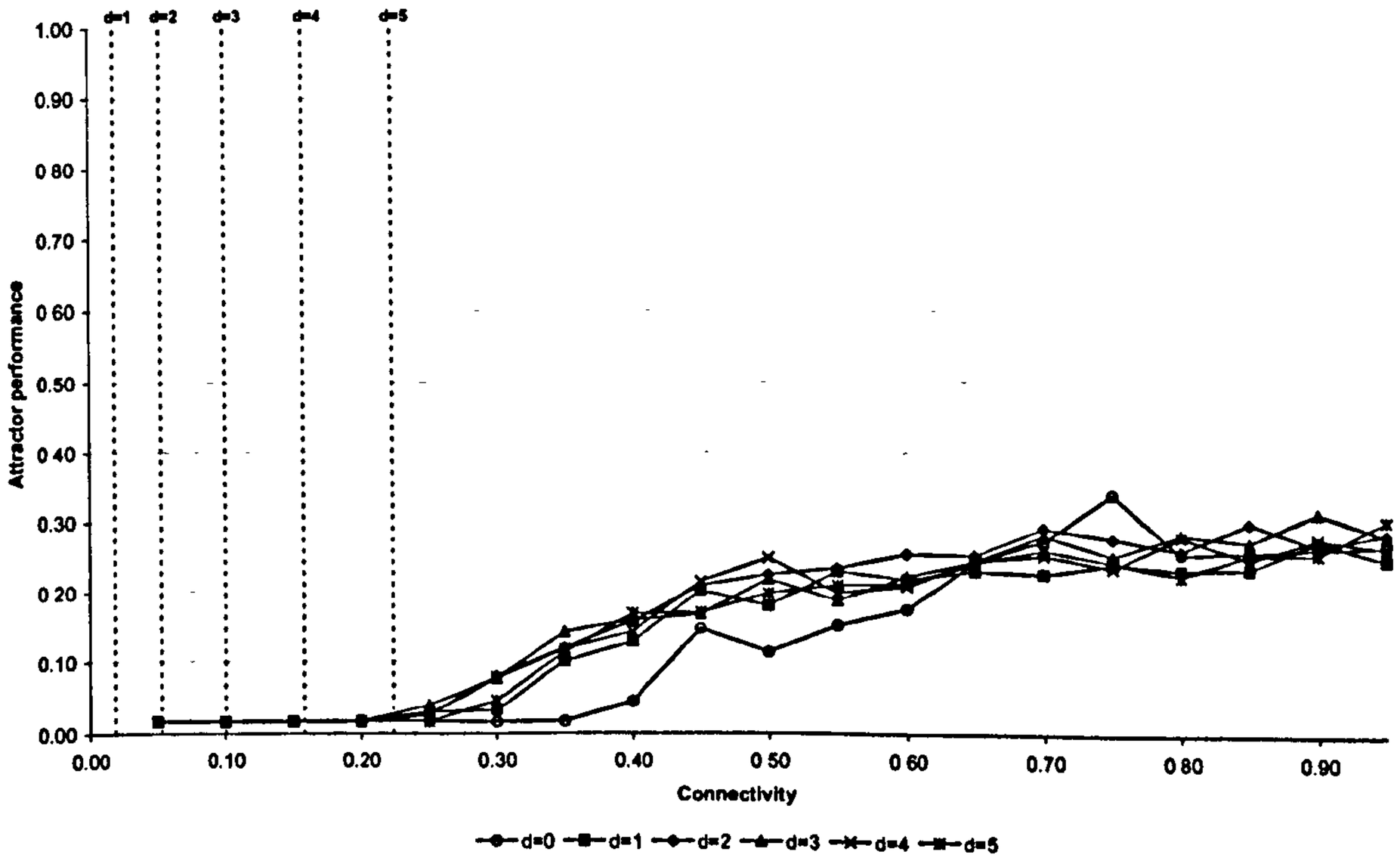


Figure 9.13: The attractor performance of networks initially connected using the local neighbourhood strategy and with further symmetric connectivity added at random. Training patterns were from the character data set. Attractor performance is shown at 5% connectivity intervals. The dashed vertical lines represent the level of neighbourhood connectivity before either compensatory or additional random connectivity was added. The loading on the network is =0.1250 (50 patterns).

Figure 9.13 demonstrates the same effect as was seen for higher loadings of geometric data. The networks exhibit much the same performance regardless of the initial level of neighbourhood connectivity. The combination of compensatory and extra random connectivity is, in all probability, again masking the initial local connectivity.

The point at which the networks begin to exhibit non-trivial attractor performance is, in all cases barring d=0, between connectivity levels of 0.20 and 0.30. The maximum attractor performance achieved by any network is approximately R=0.34 when using purely random connectivity.

Crucially, it can now be seen that a benefit in attractor performance is obtained by establishing the initial level of connectivity using a local neighbourhood strategy. While there appears to be no great benefit in extending the local neighbourhood beyond d=1, having some local connectivity does provide better attractor performance than is produced from purely random connectivity.

It is probable that the benefit of local connectivity was not apparent at the lower loading of α=0.0125 (5 patterns) because the small number of patterns was easily learnt regardless of the connectivity topology.

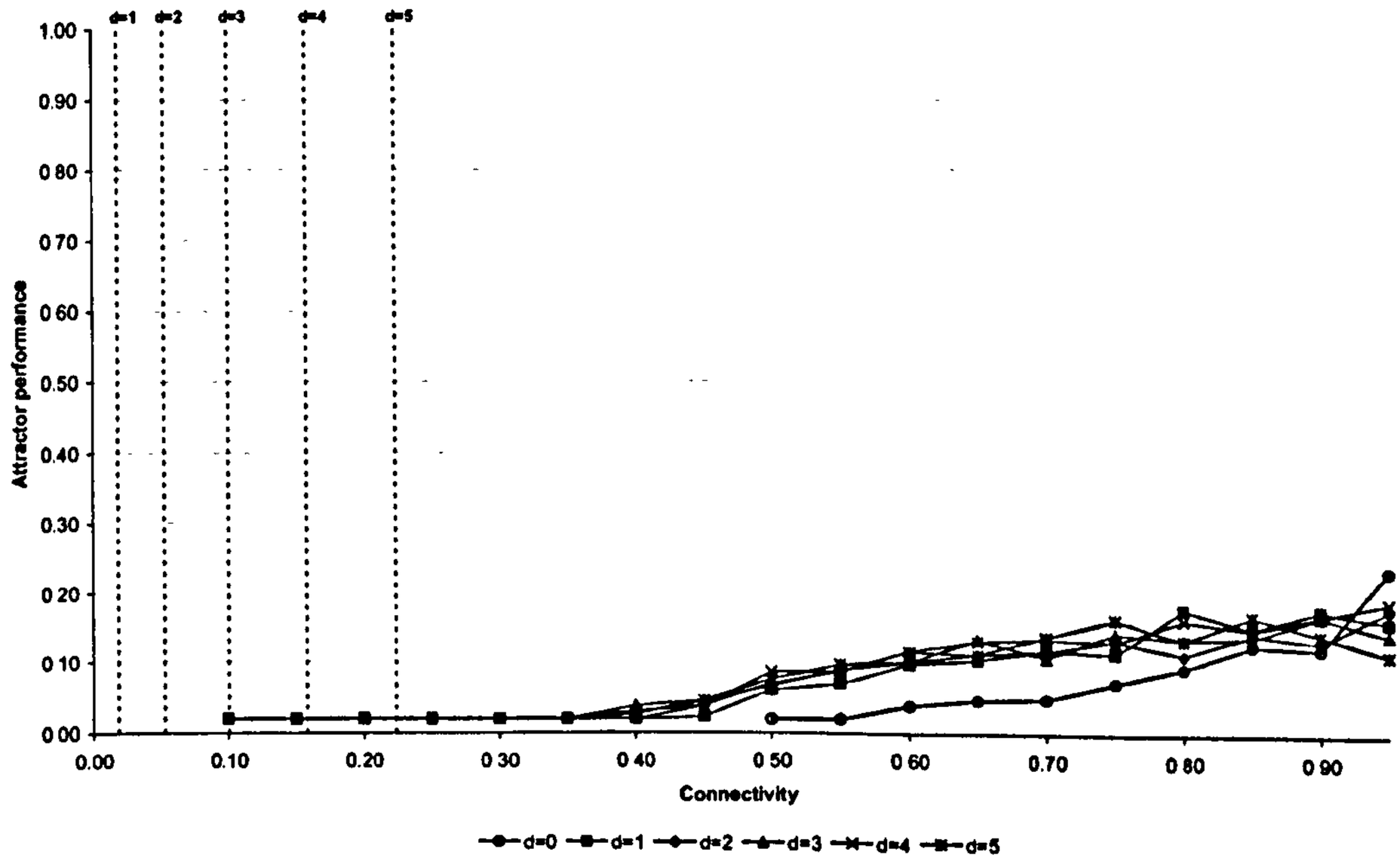


Figure 9.14: The attractor performance of networks initially connected using the local neighbourhood strategy and with further symmetric connectivity added at random. Training patterns were from the character data set. Attractor performance is shown at 5% connectivity intervals. The dashed vertical lines represent the level of neighbourhood connectivity before either compensatory or additional random connectivity was added. The loading on the network is •=0.2500 (100 patterns).

Figure 9.14 is as the previous graphs but shows results for networks with a pattern load of α=0.2500 (100 patterns). The poorer performance of randomly connected networks is clearly shown with the line representing d=0 falling below those representing networks with degrees of local connectivity. It is interesting, though expected, that the attractor performance of the randomly connected network rises to meet that of the partially locally connected networks as the level of connectivity in all the networks approaches maximum.

9.5. Summary and Conclusions

This chapter has presented the results of two sets of experimental work. The first set of experiments was designed to examine the degree of compensatory connectivity required by networks in which the initial connectivity was established using a local neighbourhood strategy. The compensatory connectivity was used to allow neurons which had failed to correctly classify their input patterns during an earlier phase of training to now classify them correctly. Once the networks had been compensated, they were analysed with respect to their storage efficiency and attractor performance.

Regardless of the type of data being learnt, the greatest amount of compensatory connectivity required was for networks with neighbourhoods of sizes d=1 and d=2. At no point however was the amount of compensatory connectivity enough to increase total connectivity above the level of pre-compensation connectivity in networks with neighbourhoods of size d=3.

It is clear however, that for both geometric and character training patterns, the local connectivity is permitting capacities well in excess of the notional maximum of $2N$. This is only the case though for networks with local neighbourhoods of sizes d=1, d=2, and d=3. These are the ranges at which local correlation both within and between the training patterns were shown to be greatest (c.f. chapter 7). These experimental results confirm the theoretical predictions of Lopez et al. (1995) described in §5.2. It should, however, be noted that in the case of networks with neighbourhoods of size d=1, this improvement in capacity comes at the expense of a large amount of compensatory connectivity which is necessarily tailored to the dataset being trained upon. This should not detract from the excellent capacities also seen in the case of networks with neighbourhoods of sizes d=2 and d=3 where the amount of compensatory connectivity required was negligible.

The attractor performance of the networks immediately post-stabilisation of the failed neurons was as expected. At low loadings, where the networks with large local neighbourhoods require no compensation, some non-trivial attractor performance exists. For any reasonable pattern load however, the attractor performance is, for all practical purposes, non-existent

The final set of results presented from the first phase of experiments showed the number of compensation/training phases required for various pattern loads. As

would be expected, the most phases were required when neighbourhood connectivity was at its minimum of d=1 and the loading was at its greatest, α=0.2500 (100 patterns). Correspondingly, the fewest training phases were required with large local neighbourhoods of d=5 and few training patterns, α=0.0125 (5 patterns). This was again true for both types of training data, geometric and character.

The second set of experiments was designed to investigate the way in which the networks' attractor performance might be enhanced post-compensation by adding additional random connectivity to each network. This was done in an attempt to counter the poor attractor performance exhibited by the just-compensated networks.

For either data type, a loading of α=0.0125 (5 patterns) does not appear to be enough for local connectivity to provide any particular advantage in terms of attractor performance. As the initial neighbourhood connectivity increases, the attractor performance at any given level of connectivity decreases. The probable cause of this is the proportion of random connectivity. For example, a network with initial neighbourhood connectivity at d=1 has a pre-compensation level of connectivity equal to approximately 0.02 of potential maximum connectivity. A network with neighbourhoods of size d=3 has a pre-compensation level of connectivity at around 0.1 of maximum. If the total connectivity after compensatory and supplementary connections have been added is 0.15, it is clear that random connectivity must form a greater proportion of that in the case of network with neighbourhoods of size d=1. This further supports the idea the some non-random connectivity may be important for good attractor performance.

Furthermore, at any particular level of connectivity, the attractor performance of the networks learning character data appears to be higher than that of those learning geometric data at a loading of α=0.0125 (5 patterns). This can be explained using Gardner's (1988) hypothesis that biased patterns should lead to higher attractor performance and is supported by the work of Davey and Hunt (2000) in which this is experimentally shown to be the case.

At a pattern load of α=0.1250 (50 patterns), it becomes much harder to distinguish between the performances of each of the locally connected networks. Apart from the poorer maximum attractor performance in the case of networks trained on

character data, one clear difference exists between the two sets of networks. For networks trained on geometric data, the attractor performance for networks with purely random connectivity (d=0) is consistently higher than for those where some of the connectivity has been established locally. For networks trained on character data it can be seen that, between levels of connectivity of 0.35 and 0.65, the attractor performance of the randomly connected networks is below that of any of those with a degree of local neighbourhood connectivity.

At a loading of α=0.2500 (100 patterns), there is a difference in performance between networks with only random connectivity and those with some neighbourhood connectivity. For the character data, the neighbourhood connectivity provides clearly superior attractor performance and the geometric data shows a tangible but less clear advantage. The difference in the two datasets can be attributed to the differing levels of bias in the data.

The experiments in this chapter have shown that there is a demonstrable benefit to establishing some of the connectivity locally before compensating failed neurons as per the scheme described in this chapter. It has been shown that, in the case of the networks investigated in this work, only a small amount of local connectivity is required in order for this benefit to be seen.

The performance improvement has only been shown to exist between particular connectivity ranges and when the pattern loading is fairly high. The effect was shown to be greater in the case of the character-based training data.

10. CONCLUSIONS

10.1. Introduction

The purpose of this final section is to draw attention to the conclusions that may be inferred from this body of work. Practical implications of this work are also discussed. Finally, suggestions for directions in which future, derived work might be taken are given.

10.2. Summary of Achievements

Through performing this investigation I have produced a number of clearly identifiable achievements. I have:

- **Investigated variants of the Hopfield network with specific focus on high performance learning rules.**

  In chapter 2, I presented a review of the field of Hopfield-type associative memories. I demonstrated that alongside the standard Hopfield learning rule based on Hebbian principles, a number of high performance learning rules exist which provide higher capacity and stronger attractor performance. I classified the resulting networks in accordance with the categories suggested by Abbott (1990).

- **Comparatively evaluated the learning rules presented in chapter 2.**

  Using a number of the performance metrics outlined in chapter 3, I carried out a comparative evaluation of the learning rules presented in chapter 2. The performance metrics were used to determine which of the rules might be most suitable for use in later work. Based on the results of the performance metrics I concluded that the most suitable rule was Gardner's Symmetric Local Learning algorithm (1988).

  I presented the results of this work at ICAANGA 2001 in Prague, Czech Republic (Turvey, Hunt et al., 2001).

- **Investigated existing measures and analysis tools and proposed a novel network performance metric.**

  In chapter 3, I presented an introduction to a number of common performance metrics suitable for the analysis of Hopfield-type associative memories. Two modifications to an existing attractor performance measure, that of Kanter and

Sompolinksy (1987), were proposed to address inconsistencies in the calculation of the original measure.

I proposed a new measure that provides a more intuitive evaluation of the attractor performance of a network. The new measure permits a comprehensive analysis of the size and shape of the attractor basins of a network's stored patterns on an individual basis while maintaining the ability to produce a comparable measure for a network as a whole.

- **Produced a suite of investigative tools for the purpose of studying the performance of associative memory architectures.**

  During the course of this investigation I developed an extensible neural network simulator that permitted the easy addition of new learning rules and analysis tools. The flexibility of the simulator's architecture makes it readily adaptable to future avenues of research in this area.

- **Investigated the intra- and inter-pattern correlation of a selection of non-random training data.**

  In Chapter 7, I presented the results of an investigation into the structural nature of two sets of artificially generated non-random training data. I showed that, in non-random training data, significant levels of local correlation existed when measured both within and between training patterns. The variability in local correlation between two different data sets was demonstrated.

  Additionally, information was gathered as to the level of site activity present in the training patterns; information that might potentially be useful in further research.

- **Empirically evaluated post-training dilution techniques.**

  In chapter 6, I examined the effect of post-training synaptic removal. The diluted networks were extensively analysed with respect to several performance metrics.

  I evaluated two post-training removal techniques. The first removed connections at random; the second used information about the values of the weights on the connections to determine those that should be removed. I termed this smallest-first dilution. The strategy providing the best performance with respect to the performance metrics used was shown to be smallest-first.

  The investigation of a non-random removal strategy in conjunction with high-performance learning rules represents novel work and I presented the results of this at RASC 2002 in Nottingham, UK (Turvey, Hunt et al., 2002).

- **Empirically evaluated two techniques for establishing sparse connectivity prior to training.**

  In chapter 8, I presented the evaluation of two strategies for establishing sparse connectivity prior to training. The first of these strategies was simple random connectivity; the second, a technique whereby individual neurons were connected to others in their local neighbourhoods. This work was based on the proposal of Garner (1988) who suggested that connectivity might be established at ranges corresponding to those at which strong local correlation was observed in the training patterns. The work of Lopez et al. (1995) gave further indication as to specific benefits this approach might provide.

  I demonstrated that particular benefits were possible when combining local connectivity with locally correlated training patterns. This work forms the key novel aspect of this thesis.

- **Proposed, modelled, and evaluated a technique for establishing structured connectivity.**

  Building on the work presented in chapter 8, in chapter 9 I examined one strategy by which the attractor performance of sparsely connected networks might be further improved through the addition of further connectivity.

The performance results from networks with identical levels of connectivity established with differing degrees of 'localness' were compared. I demonstrated that, under certain conditions, networks with a measure of local connectivity exhibited better attractor performance than did those with only random connectivity.

This crucial result corresponds with the hypothesis and justification of the approach taken that was set out in chapter 5.

10.3. Practical Implications

While it was stated at the outset (c.f. chapter 1) that biological plausibility was not a primary goal of this work, it was suggested that, where practicable, obvious implausibility might be avoided. For example, features of the final networks such as symmetric connectivity, while biologically implausible, were introduced for practical reasons. Sparse connectivity however, has been a key concern throughout the investigation.

Sparse connectivity also has implications for the implementation of associative memory neural networks in hardware. While it is acknowledged that a degree of random connectivity has been added to the networks investigated in chapter 9, it was shown that a quantity of the overall connectivity could be beneficially established locally. Overall, sparsely connected networks should be easier to implement in hardware than fully-connected ones due to the reduced physical costs.

10.4. Future Work

While this investigation has not arrived at any definite heuristic for the construction of structured, sparse connectivity, it has raised several interesting questions. To conclude this thesis, some potential avenues for future work are:

Asymmetric connectivity: symmetric connectivity was chosen as a key constraint based on it being one of Hopfield's three requirements for the existence of point attractors (Hopfield, 1982) and a desire to kept the network update dynamics as simple as possible. The biological implausibility of this requirement and the fact that it is somewhat wasteful of the available connectivity suggests the pursuit of mechanisms by which this constraint could be removed or relaxed might

be beneficial. Experimental work (Davey, 2003) has suggested that symmetric connectivity might not be as imperative as has been previously thought.

Scalability – larger networks: one of the motivations behind this work was stated to be a desire to make the creation of large networks (> 1,000,000 neurons) a more practical proposition. To this end, the relatively small networks employed in this work serve as a test bed and proof-of-concept for techniques that might be useful in constructing larger networks. Crucially, the work carried out in this thesis has demonstrated the success of connectivity strategies providing $O(N)$ scaling with respect to the network size rather than the $O(N^2)$ scaling that is seen in fully-connected networks.

Increasing the network size would significantly reduce the problems due to edge effects. In a network 1000 neurons square, neighbourhood sizes that are small when compared with the total size of the network will provide input patterns with large dimensionality. This has the potential to reduce the possibility of introducing the linear inseparabilities that occur when input dimensionalities are very small.

Perfect attractor performance: as the size of implemented networks increases, the impact of a single incorrect neuron on the resemblance of a recalled pattern to the originally stored pattern becomes increasingly less. Throughout the work described in this report the requirement has been for recalled patterns to exactly match the corresponding stored memory. Permitting a number of failed neurons has the potential to further increase recall ability of these networks and the manner in which the attractor performance analysis tools might be modified to take this into account may be worthy of investigation.

New compensation strategies: the technique for correcting failed neurons used in the work presented in chapter 9 was an unsophisticated one. Further investigation into techniques whereby the new incoming connections are chosen with consideration given to the information they will provide may be possible. The site activity analysis demonstrated in chapter 8 is one example of the type of information that might be taken into account. Some work of this nature has been investigated by Stiefvater et al. (1993).

Supplementary connectivity strategies: again, the strategy for adding supplementary connectivity in an effort to improve attractor performance post-compensation (c.f. chapter 9) is a naïve one. It may be possible to target additional connectivity such that the information provided by each new pre-synaptic neuron creates a maximally beneficial training set.

Abbot, L. F. (1990). "Learning in Neural Networks." Network: Computation in Neural Systems **1**: 105-122.

Abbot, L. F. and T. B. Kepler (1989). "Optimal learning in neural network memories." Journal of Physics A: Mathematical and General **22**: L711-L717.

Abbot, L. F. and T. B. Kepler (1989). "Universality in the space of interactions for network models." Journal of Physics A: Mathematical and General **22**: 2031-2038.

Amit, D. J. (1989). Modeling Brain Function: The world of attractor neural networks. Cambridge, Cambridge University Press.

Athithan, G. (1995). "A comparative study of two learning rules for associative memory." Pramana: Journal of Physics **45**(6): 569-582.

Blatt, M. G. and E. G. Vergini (1991). "Neural Networks: A Local Learning Prescription for Arbitrary Correlated Patterns." Physical Review Letters **66**(13): 1793-1796.

Bouten, M. (1990). "Storage Capacity of Diluted Neural Networks." Lecture Notes in Physics **368**: 225-236.

Bouten, M., A. Engel, et al. (1990). "Quenched versus annealed dilution in neural networks." Journal of Physics A: Mathematical and General **23**: 4643-4657.

Buckingham, J. and D. J. Willshaw (1993). "On setting unit thresholds in an incompletely connected associative net." Network: Computation in Neural Systems **4**: 441-459.

Caelli, T., L. Guan, et al. (1999). "Modularity in Neural Compution." Proceedings of the IEEE **87**(9): 1497-1518.

Canning, A. and E. Gardner (1988). "Partially connected models of neural networks." Journal of Physics A: Mathematical and General **21**: 3275-3284.

Changeux, J. P. and A. Danchin (1976). "Selective stabilization of developing synapses as a mechanism for the specification of neural networks." Nature **264**: 705-712.

Chechik, G., I. Meilijson, et al. (1998). "Synaptic Pruning in Development: A Computational Account." Neural Computation **10**: 1759-1777.

Cover, T. M. (1965). "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition." IEEE Transactions on Electronic Computers **EC-14**: 326-334.

da Silva, C. R., F. A. Tamarit, et al. (1995). "Generalization in a diluted neural network." Journal of Physics A: Mathematical and General **28**: 1593-1602.

Davey, N. (2003). "Sign-constrained high capacity associative memory models." Unpublished manuscript.

Davey, N. and S. P. Hunt (2000). A Comparative Analysis of High Performance Associative Memory Models. Proceedings of the 2nd International ICSC Symposium on Neural Computation (NC'2000).

Davey, N., S. P. Hunt, et al. (2002). "High Capacity Recurrent Associative Memories." Unpublished manuscript.

Derrida, B. (1989). "Distribution of the activities in a diluted neural network." Journal of Physics A: Mathematical and General **22**: 2069-2080.

Derrida, B., E. Gardner, et al. (1987). "An Exactly Solvable Asymmetric Neural Network Model." Europhysics Letters 4(2): 167-173.

Diederich, S. and M. Opper (1987). "Learning of Correlated Patterns in Spin-Glass Networks by Local Learning Rules." Physical Review Letters **58**(9): 949-951.

Evans, M. R. (1989). "Random dilution in a neural network for biased patterns." Journal of Physics A: Mathematical and General **22**: 2103-2118.

Forrest, B. M. (1988). "Content-addressability and learning in neural networks." Journal of Physics A: Mathematical and General **21**: 245-255.

Gardner, E. (1988). "The space of interactions in neural network models." Journal of Physics A: Mathematical and General **21**: 257-270.

Gardner, E., H. Gutfreund, et al. (1989). "The phase space of interactions in neural networks with definite symmetry." Journal of Physics A: Mathematical and General **22**: 1995-2008.

Gardner-Medwin, A. R. (1976). "The Recall of Events through the Learning of Associations between their Parts." Proceedings of the Royal Society of London. Series B, Biological Sciences **194**(1116): 375-402.

Greenfield, P. M. (1991). "Language, tools, and brain: The ontogeny and phylogeny of hierarchically organized sequentional behaviour." Behavioural and Brain Sciences **14**: 531-595.

Gurney, K. (1997). An Introduction to Neural Networks. London, UCL Press.

Haykin, S. (1999). Neural networks: a comprehensive foundation. Upper Saddle River, NJ, Prentice-Hall.

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

Hertz, J., A. Krogh, et al. (1991). Introduction to the Theory of Neural Computation. Redwood City, CA, Addison-Wesley Publishing Company.

Hinton, G. E. and T. J. Sejnowski (1983). Learning and relearning in Boltzmann machines. Parallel Distributed Processing: Explorations in Microstructure of Cognition. D. E. Rumelhart and J. L. McClelland. Cambridge, MA, MIT Press.

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities." Proceedings of the National Academy of Sciences of the United States of America - Biological Sciences **79**: 2554-2558.

Hopfield, J. J. (1984). "Neurons with graded response have collective computational properties like those of two-state neurons." Proceedings of the National Academy of Sciences of the United States of America - Biological Sciences **81**: 3088-3092.

Jacobs, R. A. and M. I. Jordan (1992). "Computational Consequences of a Bias toward Short Connections." Journal of Cognitive Neuroscience 4(4): 323-336.

Kanter, I. and H. Sompolinsky (1987). "Associative recall of memory without errors." Physical Review A **35**(1): 380-392.

Karlholm, J. M. (1993). "Associative Memories with Short-Range, Higher Order Couplings." Neural Networks **6**: 409-421.

Kepler, T. B. and L. F. Abbot (1988). "Domains of attraction in neural networks." Journal of Physics: France **49**: 1657-1662.

Kinzel, W. (1987). "Spin Glasses and Memory." Physica Scripta **35**: 398-401.

Kohonen, T. and M. Rouhonen (1973). "Representation of associated data by matrix operators." IEEE Transactions on Computers **22**: 701.

Komoda, A., R. Serneels, et al. (1991). "Robustness against random dilution in attractor neural networks." Journal of Physics A: Mathematical and General **24**: L743-L749.

Kothari, R. and R. Lotlikar (1997). Effect of Pruning Small Weights in Correlation Associative Memories. IEEE International Conference on Neural Networks, Texas, USA.

Krauth, W. and M. Mezard (1987). "Learning algorithms with optimal stability in neural networks." Journal of Physics A: Mathematical and General **20**: L745-L752.

Krebs, P. R. and W. K. Theumann (1999). "Categorization in the symmetrically dilute Hopfield network." Physical Review E: Statistical, Nonlinear, and Soft Matter Physics **60**(4): 4580-4587.

Levy, N., D. Horn, et al. (1999). "Associative Memory in a Multi-modular Network." Neural Computation **11**: 1717-1737.

Little, W. A. (1974). "The existence of persistent states in the brain." Mathematical Biosciences **19**: 101-120.

Lopez, B., M. Schroder, et al. (1995). "Storage of correlated patterns in a perceptron." Journal of Physics A: Mathematical and General 28: L447-L452.

Mallot, H. A. and W. von Seelen (1989). Why cortices? Neural networks for visual information processing. Visuomotor Coordination: Amphibians, Comparisons, Models and Robots. J.-P. Ewert and M. Arbib. New York, Plenum Press: 357-382.

Marr, D. (1971). "Simple memory: a theory for archicortex." Philosophical Transactions of The Royal Society, London **176**: 23-81.

McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity." Bulletin of Mathematical Biophysics **5**: 115-133.

Morita, M. (1993). "Associative Memory with Nonmonotone Dynamics." Neural Networks **6**: 115-126.

Muller, B. and J. Reinhardt (1991). Neural Networks: An Introduction. Berlin, Springer-Verlag.

Müller, K.-R., T. Stiefvater, et al. (1993). Associative Storage and Retrieval of Highly Correlated Natural Pattern Sets in Diluted Hopfield Networks. IEEE International Conference on Neural Networks, San Francisco, USA.

O'Kane, D. and A. Treves (1992). "Short- and long-range connections in autoassociative memory." Journal of Physics A: Mathematical and General **25**: 5055-5069.

O'Leary, D. D. M. (1989). "Do cortical areas emerge from a protocortex?" Trends in Neurosciences **12**: 400-406.

Personnaz, L., I. Guyon, et al. (1986). "Collective computational properties of neural networks: New learning mechanisms." Physical Review A: Atomic, Molecular, and Optical Physics **34**(5): 4217-4228.

Plakhov, A. Y. and S. A. Semenov (1994). "Neural networks: iterative unlearning algorithm converging to the projector rule matrix." Journal de Physique France **4**: 253-260.

Rosenblatt, F. (1958). "The Perceptron: A probabilistic model for information storage and organization in the brain." Psychological Review **65**: 386-408.

Ruppin, E. (1995). "Neural Modelling of Psychiatric Disorders." Network: Computation in Neural Systems **6**(4): 635-656.

Ruppin, E. and J. A. Reggia (1995). "Patterns of Functional Damage in Neural Network Models of Associative Memory." Neural Computation **7**: 1105-1127.

Ruppin, E. and J. A. Reggia (1998). "Seeking order in disorder: computational studies of neurologic and psychiatric diseases." Artificial Intelligence in Medicine **13**: 1-12.

Schultz, A. (1995). "Five Variations of Hopfield Associative Memory." Journal of Artificial Neural Networks **2**(3): 285-294.

Sompolinsky, H. (1986). "Neural networks with nonlinear synapses and a static noise." Physical Review A: Atomic, Molecular, and Optical Physics **34**(3): 2571-2574.

Stent, G. S. (1973). "A Physiological Mechanism for Hebb's Postulate of Learning." Proceedings of the National Academy of Sciences of the United States of America **70**: 997-1001.

Stiefvater, T., K.-R. Müller, et al. (1993). "Sparsely Connected Hopfield Networks for the Recognition of Correlated Pattern Sets." Network: Computation in Neural Systems 4: 313-336.

Storkey, A. J. (1997). "Increasing the capacity of a Hopfield network without sacrificing functionality." Proceedings of the International Conference on Artificial Neural Networks: 451-456.

Storkey, A. J. and A. Valabregue (1999). "The basins of attraction of a new Hopfield learning rule." Neural Networks **12**: 869-876.

Sutton, J. P., J. S. Beis, et al. (1988). "Hierarchical model of memory and memory loss." Journal of Physics A: Mathematical and General **31**: 4443-4454.

Tsodyks, M. V. (1989). "Associative memory in neural networks with the Hebbian learning rule." Modern Physics Letters B **3**(7): 555-560.

Tsodyks, M. V. and M. V. Feigel'man (1988). "The enhanced storage capacity in neural networks with low activity level." Europhysics Letters **6**: 101-105.

Turvey, S. P., S. P. Hunt, et al. (2001). "An Experimental Assessment of the Performance of Several Associative Memory Models." Proceedings of 5th International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA): 70-73.

Turvey, S. P., S. P. Hunt, et al. (2002). "Non-Random Weight Dilution in High Performance Associative Memories." Proceedings of 4th International Conference on Recent Advances in Soft Computing: 37-42.

Vishwanathan, R. R. (1995). "Fault tolerance in simple perceptrons." Physics Letters A 188: 55.

Vishwanathan, R. R. (1995). "Synapse removal in discrete neural networks." Journal of Physics A: Mathematical and General 28: L25-31.

Willshaw, D. J., O. P. Buneman, et al. (1969). "Non-Holographic Associative Memory." Nature (London) **222**: 960-962.

Appendices

A. DEVELOPING AN ASSOCIATIVE MEMORY SIMULATOR

A.1. Overview

In order to carry out the series of experiments that were required for this work it was necessary to develop software for the purpose of simulating the architectures and learning rules being used. This section endeavours to give an overview of the requirements of such a simulator and an insight into some of the factors that had to be considered in its design and implementation.

In total, three simulators were built, each evolving from the last as deficiencies were found or requirements outstripped capabilities. The first of these was built around a very simple monolithic design. The purpose of this first attempt was more about familiarisation with the problems of constructing such a simulator than about building something genuinely useful. It was, if you will, a prototype designed to fulfil the maxim that "you never really understand a problem until you've attempted a solution".

The second simulator was significantly more complex the first. It was highly parameterised and flexible and is discussed in some detail below. This simulator was, however, not without its flaws and eventually gave way to the third and final version which, while not perfect, served its purpose well. It is this simulator that is the subject of examination in the latter part of this section.

All three of the simulators were developed in Java. The rationale for this was that the small sacrifice in terms of speed of execution would be offset by the advantages brought by the cross-platform capabilities of Java.

A.2. A Simple Associative Memory Simulator

The first simulator implementation was a simple monolithic design. Parameters to the simulator were hard-coded into the program source. Each learning rule implemented was represented by a new program. While this had the advantage of high speed and simplicity of design, there was the huge disadvantage of being unable to modify simulation parameters without recompilation of the entire program.

A.3. Simulator 2: A First Stab at Flexibility

It was clear from the start that the monolithic prototype simulator was going to be unsuitable for running the gamut of learning rules and analyses that would be required for this project. Parameterising the simulator was going to be essential if it was ever to be flexible enough to run multiple experimental runs without significant intervention or alteration each time.

A.3.1. Design and Implementation

Strictly speaking, the second simulator was also monolithic though an attempt was made to separate out different areas of functionality into distinct classes/files. The core of the simulator is formed by the *NeuralNetwork* class which takes responsibility for managing a network's weights, state, and training set. 'Running' the network as per the update dynamics is also handled by this class.

Rather than being an entity in its own right, a new learning rule is considered to be a type of network and so inherits from the base *NeuralNetwork* class. This enables learning rules to directly access and manipulate both the weights and the training set associated with the network. These learning rule/network classes are dynamically loaded as required according to the command line parameters.

Analysis tools are kept separate in their own class and operate on, rather than form part of, network objects.

Finally, the entire system is controlled by a class which is responsible for instantiating a network and executing analysis tools as specified by parameters which may be passed to it on the command line.

A.3.2. Advantages and Disadvantages

The parameterisation which initially was an asset and made the simulator extensible later became a problem. The requirement that each new parameter identifier be hard-coded into the simulator meant that the entire system needed recompiling for what were often relatively minor changes. Similarly, when a new learning rule was added to the system its name also had to be hard-coded in a list of network models that were available for selection via the command line parameters. The class representing the model named was dynamically loaded at runtime and made available within the Java namespace.

Over time, the number of parameter identifiers required in order to provide all the information necessary to all of the simulator's functionality became unwieldy and difficult to manage. Compounding this was the fact that not all simulations followed the same pattern of execution; some produced values which were means of multiple runs while others simply returned a Boolean true or false.

Eventually, managing the simulator was taking up more time than it was economical to invest.

Nothing written above should detract from the positive aspects of both the experience of implementing the simulator and the eventual product. The parameterisation *was* initially an asset and much time was saved compared with the alternative of coding individual single-use programs. The dynamic class loading infrastructure paved the way for what was to become a full-blown system of plug-in extensions in the final simulator and much was learnt about the most appropriate way in which to permit scripting or parameterisation of the software.

A.4. NetSim: a Flexible Associative Memory Simulation Architecture

The deficiencies in the second simulator having become intolerable, work began on developing the next generation of the software. From the beginning it was designed to be both fully scriptable and extensible. This enabled the final move away from having to recompile the entire simulator should a new analysis tool be developed or a extra parameter required for a new learning rule.

In the manner in which the very first simulator had acted as a prototype or scratchpad for the technologies incorporated in the second, so the second acted likewise for the technologies and features built into the third. The mechanism by which learning rules, modelled as independent architectures built upon a generic network framework, could be dynamically loaded at run-time evolved into the plug-in system that provides the new simulator with the majority of its power with respect to extensibility without recompilation. In a similar manner, the parameterisation system of the second simulator and the flaws within it gave rise to the need for and development of the new scripting engine.

The third simulator was also the first with a specific name, NetSim.

A.4.1. Benefits of an Object-Oriented Approach

Traditionally, an object-orientated approach has been regarded as being unsuitable for the design of neural network simulators. The level of interaction required between very large numbers of components is such that the overheads of communication between objects are likely to be significant. Nevertheless, the inherent structure of associative memory neural networks lends itself very naturally to being modelled as classes of objects. The requirement for ease of extension and maintainability adds further weight to the merit of such an approach.

A.4.2. Design and Implementation

Any discussion of the operation of the NetSim simulator must begin with the scripting engine since it is this that specifies the run-time environment and controls the running of any learning rules, tools, and utilities.

The language of choice for scripting NetSim was the Extensible Mark-up Language (XML). The simplicity with which one may develop a configuration/scripting

language within XML belies the full power of XML itself. This ease of use, coupled with native support for XML within Java, made it the only sensible option.

A discussion of XML itself is beyond the scope of this document but further information, if required, may be found at http://www.w3.org/XML/.

As XML permits the creation of arbitrary tags it was straightforward to specify control blocks would hold the parameters for specific areas of simulator functionality.

In order to develop an understanding of the flexibility of the NetSim software it is important that the scripting engine itself is understood. All other functions flow from what is specified within the NetSim configuration file.

Each control block takes, as its first parameter, the name of a plug-in class which will provide the necessary functionality. For instance, some names of plug-in classes which can be used in the *LearningRule* control block are: *BipolarHopfield*, *ILL* (Iterative Local Learning), *BV* (Blatt & Vergini). When the configuration file parser encounters a plug-in name it loads and links it dynamically from the file system.

The control blocks available are:

Controller : This is the main control block. Plug-ins specified here are responsible for managing the creation of networks, the execution of learning rules, and the performing of any analyses on the resulting weight matrices. Also delegated to these plug-ins is the creation of training sets. In some cases training data is randomly generated according to specified parameters; in others, it is loaded from data files on disk.

Network : This control block has no corresponding plug-ins. It's purpose is to allow parameters general to all networks to be defined. Currently these parameters are limited to specifying the dimensions of the network.

ConnectionStrategy : Plug-ins specified by this control block control the pattern of connectivity of the network. Examples of these are the plug-ins *AlwaysConnect*, *RandomlyConnect*, and *NeighbourhoodConnect2D*.

InputFunction : This control block specifies the plug-in that will provide the input function for each neuron in the network. The input function will determine the activation value of the neuron. Currently, the only input function plug-in is *StandardLWS* which computes the linear weighted sum of a neuron's inputs.

OutputFunction : The output function control block specifies the plug-in implementing the output function for each neuron. Using the activation value obtained from the input function the plug-in calculates a neuron's output value. Currently, only the plug-in *BipolarStep* is available which implements a bipolar step function.

UpdateRule : The update rule control block specifies the plug-in implementing the network's update dynamics. Currently these include *RandomAsync* (random asynchronous updates), *RandomAsyncReplace* (random asynchronous updates with replacement), and *PseudoRandomAsync* (a high performance implementation of random asynchronous updating employing a pre-generated lookup table of random numbers).

LearningRule : This control block determines the learning rule plug-in to be used in training the network. Plug-ins exist for most of the learning rules examined in §2.5.

PreTraining : This is the first control block for which multiple instances are permitted. Plug-ins specified by this block are executed prior to the network being trained. Certain analysis tools, such as those that operate on the training set, might be run at this point or the display of certain network information might be handled by these plug-ins. Examples of these plug-ins are: *CheckConnectionSymmetry* (an analysis of the pattern of connectivity to ensure symmetry) or *TrainingSetBias* (an analysis of the tendency of the training set towards a particular value).

PostTraining : Multiple instances of this control block are also permitted. Specified within are plug-ins designed to operate after the network has been trained. For this reason most plug-ins specified here perform some type of analysis of the network's weight matrix. Example are: *AllStable* (test that all training patterns have been successfully learnt), *DisplayGammas* (an calculation of the weight matrix's gamma

values, and *KanterSompolinsky* (an analysis of the consistency of size and shape of the basins of attraction of the network)

To fulfil the requirement for extensibility without recompilation it was necessary to devise a system whereby parameters could be passed to new plug-ins without having to check for their validity within the configuration file parser. To this end, it was decided that responsibility for parameter validation would rest with the plug-ins themselves; the configuration file parser would simply pass the parameter name/value pairs to the plug-in named in the control block and ask whether or not that pair should be considered valid.

A.4.3. Advantages and Disadvantages

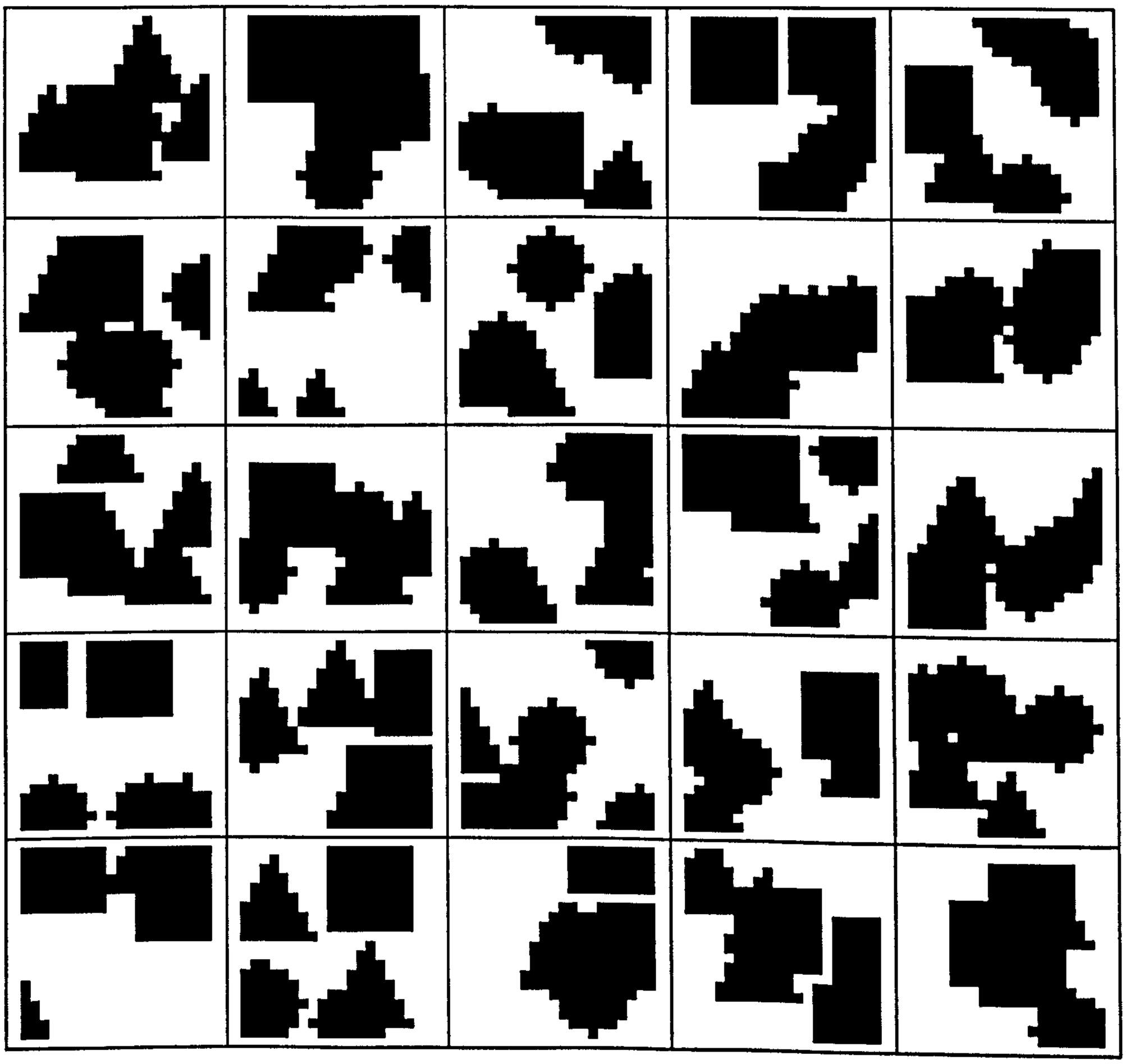
The key advantage of the NetSim simulator is the sheer flexibility and extensibility of the architecture. New learning rules, analysis tools, input functions, and output functions may all be specified as plug-ins to be loaded at run time and used as specified in the main configuration file.

The increased modularisation of the architecture lead to great benefits in the maintainability of the simulator's source code. Once the framework was in position for permit the creation and loading of plug-in modules, the core infrastructure code rarely needed modification.

The only major disadvantage of the NetSim simulator occurred when a pattern of experimentation did not quite fit the model of execution that the simulator was designed around. As with all software projects, constraints have to be established as to a program's domain of operation. Occasionally, experiments were required to be run that strayed outside of the established framework. It is thanks to the maintainability of the core code, as a result of the modularisation, that adding the extra functionality to the simulator was never particularly painful.

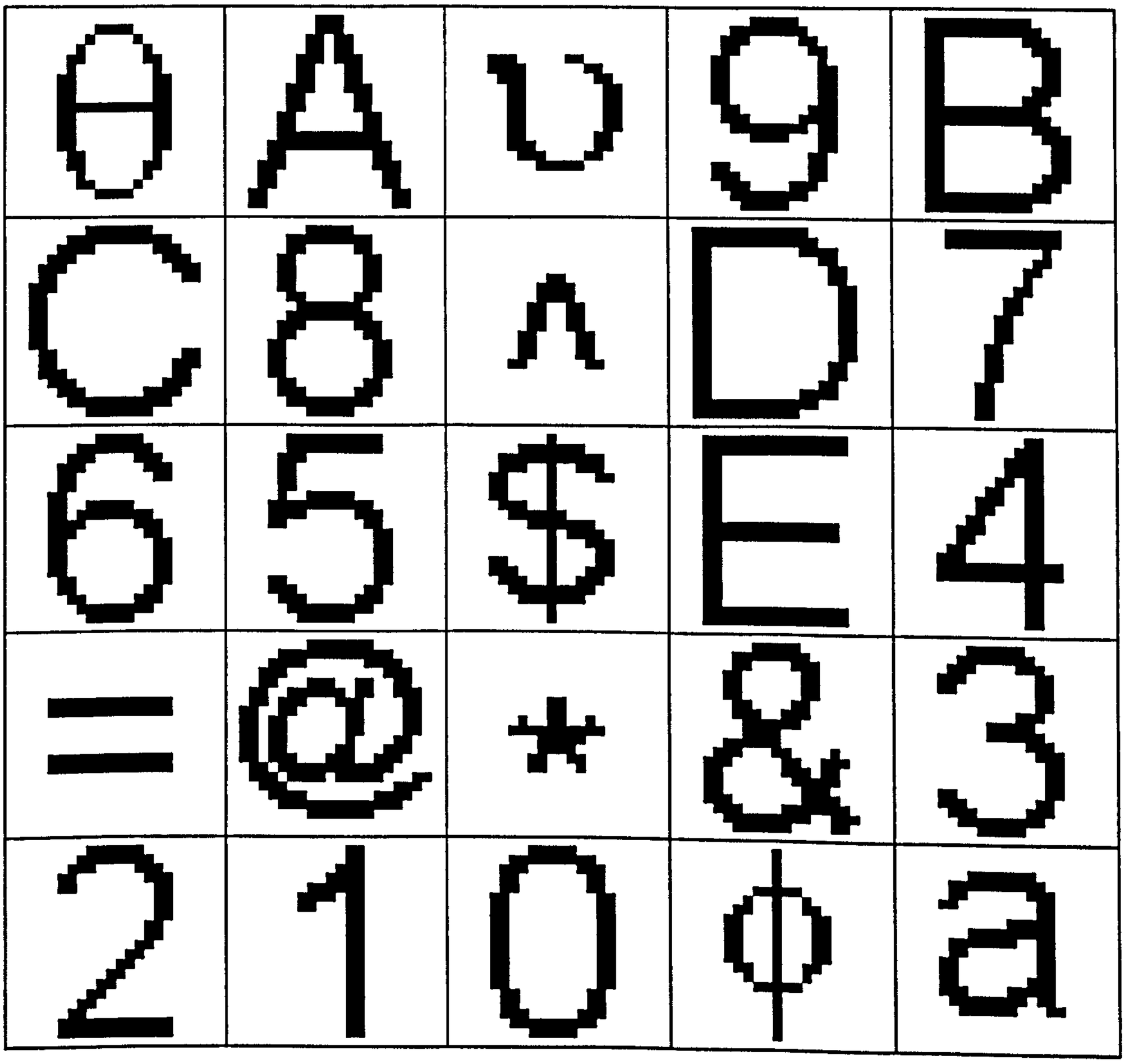
B. A SELECTION OF GEOMETRIC TRAINING DATA

The images below represent a selection of the geometric training data that was used throughout this work.



C. A SELECTION OF CHARACTER TRAINING DATA

The images below represent a selection of the character-based training data that was used throughout this work.



D. DATA TABLES – SPARSE CONNECTIVITY

This appendix contains the tables of data from which the summary tables in chapter 8 were generated. Each values represents the mean of 5 individual simulation runs.

Random data (bias=0.5)

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	-	-	57.00
0.0250	-	-	229.00
0.0375	-	-	378.20
0.0500	-	-	392 80
0.0625	-	-	398.00
0.0750	-	-	397 80
0.0875	-	-	399.40
0.1000	-	-	399.20
0.1125	-	-	399.80
0.1250	-	-	399.40
0.1375	-	-	400.00
0.1500	-	-	400.00
0.1625	-	-	400.00
0.1750	-	-	400.00
0.1875	-	-	400.00
0.2000	-	-	400.00
0.2125	-	-	400.00
0.2250	-	-	400.00
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.1: Performance metrics for networks with 7.49 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	94.80	0.0087	-
0.0250	374.00	0.0055	-
0.0375	767.00	0.0056	1.33
0.0500	-	-	23.80
0.0625	-	-	196.40
0.0750	-	-	360.20
0.0875	-	-	395.20
0.1000	-	-	398.80
0.1125	-	-	400.00
0.1250	-	-	400.00
0.1375	-	-	400.00
0.1500	-	-	400.00
0.1625	-	-	400.00
0.1750	-	-	400.00
0.1875	-	-	400.00
0.2000	-	-	400.00
0.2125	-	-	400.00
0.2250	-	-	400.00
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.2: Performance metrics for networks with 21.09 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	31.80	0.7519	-
0.0250	57.40	0.4013	-
0.0375	118.00	0.0055	-
0.0500	198 40	0.0056	-
0.0625	288.80	0.0056	-
0.0750	640.75	0.0057	1.00
0.0875	-	-	2.80
0.1000	-	-	25.20
0.1125	-	-	198.00
0.1250	-	-	322.60
0.1375	-	-	387.80
0.1500	-	-	398 40
0.1625	-	-	399.80
0.1750	-	-	399.60
0.1875	-	-	400.00
0.2000	-	-	400.00
0.2125	-	-	400.00
0.2250	-	-	400.00
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.3: Performance metrics for networks with 39.96 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	21.20	0.7990	-
0.0250	32.40	0.7234	-
0.0375	48.80	0.5764	-
0.0500	75.00	0.1817	-
0.0625	101.20	0.0146	-
0.0750	136 00	0.0080	-
0.0875	195 80	0.0057	-
0.1000	295.60	0.0057	-
0.1125	421.20	0.0057	-
0.1250	481.20	0.0057	-
0.1375	831.20	0.0057	-
0.1500	-	-	3.20
0.1625	-	-	45.60
0.1750	-	-	211.80
0.1875	-	-	320.40
0.2000	-	-	378 40
0.2125	-	-	393.40
0.2250	-	-	398 60
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.4: Performance metrics for networks with 63 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0 0125	18.40	0.7949	-
0.0250	25 40	0.7774	-
0.0375	41.20	0.7046	-
0.0500	43.00	0.6066	-
0 0625	60.20	0 4765	-
0.0750	69.20	0.3068	-
0.0875	87.00	0.1152	-
0.1000	121.40	0.0423	-
0.1125	136.60	0.0137	-
0.1250	163.20	0.0080	-
0.1375	210.40	0.0057	-
0.1500	272.80	0.0057	-
0.1625	362.80	0.0057	-
0.1750	473.80	0.0058	-
0.1875	613.20	0.0058	-
0.2000	741.60	0.0058	-
0.2125	973.75	0.0058	1.00
0.2250	-	-	7.00
0.2375	-	-	95.60
0.2500	-	-	227.80

Table D.5: Performance metrics for networks with 89.25 MCPN

Random data (bias=0.8)

Loading	Train. Time	Attractor Perf.	Failed Units
0 0125	-	-	76.20
0.0250	-	-	217.20
0.0375	-	-	332.60
0.0500	-	-	378.00
70.0625	-	-	395.60
0.0750	-	-	397.80
0.0875	-	-	399 80
0.1000	-	-	400.00
0.1125	-	-	400.00
0.1250	-	-	400.00
0.1375	-	-	400.00
0.1500	-	-	400.00
0.1625	-	-	400.00
0.1750	-	-	400.00
0.1875	-	-	400.00
0.2000	-	-	400.00
0.2125	-	-	400.00
0.2250	-	-	400.00
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.6: Performance metrics for networks with 7.49 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	165.00	0.0082	1.50
0.0250	-	-	4.20
0.0375	-	-	9.20
0.0500	-	-	31.60
0.0625	-	-	111.20
0.0750	-	-	212.60
0.0875	-	-	311.40
0.1000	-	-	367.00
0.1125	-	-	387.40
0.1250	-	-	397.40
0.1375	-	-	397.60
0.1500	-	-	399.80
0.1625	-	-	399.60
0.1750	-	-	400.00
0.1875	-	-	399.80
0.2000	-	-	400.00
0.2125	-	-	400.00
0.2250	-	-	400.00
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.7: Performance metrics for networks with 21.09 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	65.40	0.7334	-
0.0250	150.20	0.0153	-
0.0375	212.20	0.0087	-
0.0500	512.50	0.0088	1.00
0.0625	621.80	0.0089	-
0.0750	903.67	0.0089	1.00
0.0875	969.00	0.0090	1.33
0.1000	-	-	6.20
0.1125	-	-	30.80
0.1250	-	-	92.60
0.1375	-	-	192.00
0.1500	-	-	280.00
0.1625	-	-	332.80
0.1750	-	-	369.40
0.1875	-	-	385.20
0.2000	-	-	393.00
0.2125	-	-	398.00
0.2250	-	-	399.60
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.8: Performance metrics for networks with 39.96 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	43.80	0.7611	-
0.0250	77.80	0.6987	-
0.0375	99.80	0.5696	-
0.0500	117.60	0.1423	-
0.0625	181.00	0.0158	-
0.0750	274.20	0.0089	-
0.0875	263.20	0.0090	-
0.1000	363.80	0.0090	-
0.1125	664.20	0.0091	-
0.1250	609.80	0.0091	-
0.1375	805.80	0.0091	-
0.1500	821.40	0.0091	
0.1625	983.50	0.0091	2.33
0.1750	-	-	9.20
0.1875	-	-	21.80
0.2000	-	-	55.40
0.2125	-	-	158.60
0.2250	-	-	234.80
0.2375	-	-	311.80
0.2500	-	-	343.60

Table D.9: Performance metrics for networks with 63 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	30.40	0.7841	#N/A
0.0250	51.20	0.7326	#N/A
0.0375	63.00	0.7095	#N/A
0.0500	79.20	0.6130	#N/A
0.0625	92.40	0.5641	#N/A
0.0750	110.40	0.3091	#N/A
0.0875	136.40	0.1711	#N/A
0.1000	159.40	0.0288	#N/A
0.1125	174.20	0.0144	#N/A
0.1250	219.80	0.0090	#N/A
0.1375	279.80	0.0090	#N/A
0.1500	312.20	0.0091	#N/A
0.1625	370.80	0.0091	#N/A
0.1750	396.20	0.0092	#N/A
0.1875	452.20	0.0092	#N/A
0.2000	661.00	0.0092	#N/A
0.2125	714.80	0.0092	#N/A
0.2250	863.75	0.0091	1.00
0.2375	942.75	0.0092	1.00
0.2500	#N/A	#N/A	1.60

Table D.10: Performance metrics for networks with 89.25 MCPN

Character Data

Loading	Train. Time	Attractor Perf.	Failed Units
0 0125	-	-	78.60
0.0250	-	-	190.40
0.0375	-	-	254.00
0.0500	-	-	293.80
0.0625	-	-	330.60
0.0750	-	-	332.00
0.0875	-	-	345.80
0.1000	-	-	355.60
0.1125	-	-	360.20
0.1250	-	-	363.80
0.1375	-	-	369.00
0.1500	-	-	373 00
0.1625	-	-	369.00
0.1750	-	-	381.80
0.1875	-	-	378 80
0.2000	-	-	383.20
0.2125	-	-	383 40
0.2250	-	-	382.80
0.2375	-	-	387.40
0.2500	-	-	388.00

Table D.11: Performance metrics for networks with 7.49 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	-	-	3.20
0.0250	-	-	14 60
0.0375	-	-	33.20
0.0500	-	-	75.20
0.0625	-	-	107.40
0.0750	-	-	163.00
0.0875	-	-	208.80
0.1000	-	-	237.20
0.1125	-	-	252.60
0.1250	-	-	269.40
0.1375	-	-	271.20
0.1500	-	-	284.00
0.1625	-	-	289.80
0.1750	-	-	297.00
0 1875	-	-	295.40
0.2000	-	-	307.60
0.2125	-	-	309.20
0.2250	-	-	313.00
0.2375	-	-	318.60
0.2500	-	-	320.80

Table D.12: Performance metrics for networks with 21.09 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	127.80	0.0921	-
0.0250	403.00	0.0121	1.00
0.0375	538.50	0.0140	2.00
0.0500	-	-	4.00
0.0625	-	-	8.60
0.0750	-	-	16.20
0.0875	-	-	23.80
0.1000	-	-	42 80
0.1125	-	-	66.40
0.1250	-	-	114.60
0.1375	-	-	139.80
0.1500	-	-	173.00
0.1625	-	-	198.20
0.1750	-	-	211.20
0.1875	-	-	224.40
0.2000	-	-	236.20
0.2125	-	-	243.00
0.2250	-	-	249.20
0.2375	-	-	253.00
0.2500	-	-	260.40

Table D.13: Performance metrics for networks with 39.96 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	81.20	0.6940	#N/A
0.0250	243.20	0.3480	#N/A
0.0375	309.60	0.0179	#N/A
0.0500	356 67	0.0135	1.50
0.0625	687.00	0.0134	1.50
0.0750	810.33	0.0153	2.00
0.0875	766.50	0.0158	1.67
0.1000	-	#N/A	1.60
0.1125	938.00	0.0158	3.25
0.1250	-	-	6.00
0.1375	-	-	14.20
0.1500	-	-	21.20
0.1625	-	-	38.80
0.1750	-	-	56.60
0.1875	-	-	71.00
0.2000	-	-	91.00
0.2125	-	-	118.00
0.2250	-	-	133.20
0.2375	-	-	156 80
0.2500	-	-	176.40

Table D.14: Performance metrics for networks with 63 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	68.40	0.7482	#N/A
0.0250	104.00	0.6950	#N/A
0.0375	300.00	0.3580	#N/A
0.0500	229.20	0 2246	#N/A
0.0625	293.75	0.0181	1 00
0.0750	338.50	0.0198	2.00
0.0875	607.33	0.0156	1.50
0.1000	587.00	0.0162	2 00
0.1125	757.00	0.0157	3.33
0.1250	1000 00	0.0161	3.00
0.1375	-	-	3.20
0.1500	1000 00	0.0167	3 67
0.1625	-	-	2 80
0.1750	-	-	2.80
0.1875	-	-	4 80
0.2000	-	-	5.00
0.2125	-	-	11 00
0.2250	-	-	15.60
0.2375	-	-	23.20
0.2500	-	-	32.80

Table D.15: Performance metrics for networks with 89.25 MCPN

Geometric Data

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	-	-	56.20
0.0250	-	-	197.40
0.0375	-	-	328.60
0.0500	-	-	374.20
0.0625	-	-	390 80
0.0750	-	-	394.60
0.0875	-	-	397.00
0.1000	-	-	399.00
0.1125	-	-	398.80
0.1250	-	-	398.60
0.1375	-	-	399.60
0.1500	-	-	400 00
0.1625	-	-	400.00
0.1750	-	-	400.00
0.1875	-	-	400.00
0.2000	-	-	400.00
0.2125	-	-	400.00
0.2250	-	-	400.00
0.2375	-	-	400.00
0.2500	-	-	400.00

Table D.16: Performance metrics for networks with 7.49 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	106.20	0.0109	-
0.0250	221.50	0.0080	2.00
0.0375	-	-	3.40
0.0500	-	-	15.60
0.0625	-	-	43 80
0.0750	-	-	126.20
0.0875	-	-	221.40
0.1000	-	-	272.00
0.1125	-	-	338.00
0.1250	-	-	355.60
0.1375	-	-	379.40
0 1500	-	-	384 80
0.1625	-	-	389.60
0.1750	-	-	395.20
0.1875	-	-	398 00
0.2000	-	-	397.80
0.2125	-	-	398.00
0.2250	-	-	399.60
0.2375	-	-	399.60
0.2500	-	-	399.80

Table D.17: Performance metrics for networks with 21.09 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	41.80	0.7741	-
0.0250	120.20	0.0274	-
0.0375	164 00	0.0099	-
0.0500	392.40	0.0086	-
0.0625	498.20	0.0092	-
0.0750	756.00	0.0092	1.00
0.0875	919 00	0.0091	1.00
0.1000	-	-	2.40
0.1125	-	-	4 80
0.1250	-	-	13.40
0.1375	-	-	34.60
0.1500	-	-	61.40
0.1625	-	-	106.40
0.1750	-	-	169.20
0.1875	-	-	208 80
0.2000	-	-	243 40
0.2125	-	-	271.80
0.2250	-	-	301.00
0.2375	-	-	324.20
0.2500	-	-	344.40

Table D.18: Performance metrics for networks with 39.96 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	30.40	0.7948	-
0.0250	70.20	0.7044	-
0.0375	90 00	0.5119	-
0.0500	165 40	0.0206	-
0.0625	163.60	0.0111	-
0.0750	203 40	0.0093	-
0.0875	246.80	0 0092	-
0.1000	410.60	0.0097	-
0.1125	390.20	0.0096	-
0.1250	542.20	0.0099	-
0.1375	801.20	0.0097	-
0.1500	742.40	0.0098	-
0.1625	841.00	0.0103	1.00
0.1750	1000.00	-	1.50
0.1875	-	-	2.00
0.2000	-	-	3.20
0.2125	-	-	5.80
0.2250	-	-	12.00
0.2375	-	-	16.00
0.2500	-	-	29.40

Table D.19: Performance metrics for networks with 63 MCPN

Loading	Train. Time	Attractor Perf.	Failed Units
0.0125	25 60	0.8094	-
0.0250	48 80	0.7386	-
0.0375	58 60	0.6970	-
0.0500	93 60	0.5592	-
0.0625	116.20	0.3829	-
0.0750	126.40	0.0860	-
0.0875	144.00	0.0183	-
0.1000	176.60	0.0116	-
0.1125	234 80	0.0116	-
0.1250	279.20	0.0140	-
0.1375	315.60	0.0098	-
0.1500	263.80	0.0101	-
0.1625	317.80	0 0102	-
0.1750	405.80	0.0124	-
0.1875	506.20	0.0103	-
0.2000	480.60	0.0102	-
0.2125	521.40	0.0105	-
0.2250	608.60	0 0103	-
0.2375	756.60	0.0107	-
0.2500	797.00	0.0104	-

Table D.20: Performance metrics for networks with 89.25 MCPN

E. DATA TABLES – COMPENSATORY CONNECTIVITY

This appendix contains the tables of data from which the results tables in chapter 9 were generated. Each values represents the mean of 5 individual simulation runs.

Network Stabilisation - Character data

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	7.4440	0.01	5	0.67
0.0250	7.7230	0.01	11	1.29
0.0375	8 1340	0.01	19	1.84
0.0500	8.6440	0.01	19	2.31
0.0625	9.1000	0.02	29	2.75
0.0750	9.5810	0.02	24	3.13
0.0875	10 4670	0.02	32	3.34
0.1000	10.7290	0.02	33	3.73
0.1125	11 8370	0 02	31	3.80
0.1250	12.3150	0.02	35	4.06
0.1375	13.1460	0.02	39	4.18
0.1500	14.4310	0.02	39	4.16
0.1625	15.3130	0.02	46	4.24
0.1750	16.0770	0.02	41	4.35
0.1875	17.1640	0.02	52	4.37
0.2000	17.7313	0.02	44	4.51
0.2125	19.2920	0.02	37	4.41
0.2250	20.3940	0.02	38	4.41
0.2375	22.2330	0.02	48	4.27
0.2500	22.5800	0.02	45	4.43

Table E.1: Performance metrics for networks with 7.49 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	21.0910	0.02	1	0.24
0.0250	21.1020	0.01	3	0.47
0.0375	21.1760	0.01	12	0.71
0.0500	21.1660	0.01	8	0.94
0.0625	21.2890	0.02	18	1.17
0.0750	21.3270	0.02	17	1.41
0.0875	21.3950	0.02	29	1.64
0.1000	21.4840	0.02	25	1.86
0.1125	21.6190	0.02	21	2.08
0.1250	21.7770	0.02	32	2.30
0.1375	21.6850	0.02	23	2.54
0.1500	22.0850	0.02	32	2.72
0.1625	22 0875	0.02	41	2.94
0.1750	22.2170	0.02	37	3.15
0.1875	22.4750	0.02	42	3.34
0.2000	22.6860	0.02	34	3.53
0.2125	22.7340	0.02	41	3.74
0.2250	23.0450	0.02	40	3.91
0.2375	23.3590	0.02	39	4.07
0.2500	23.5890	0.02	31	4.24

Table E.2: Performance metrics for networks with 21.09 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	39.9610	0.16	0 0000	0 13
0.0250	39.9740	0.02	0.0004	0.25
0.0375	39.9860	0.01	0.0007	0.38
0.0500	39.9660	0.01	0.0002	0.50
0.0625	40.0290	0.02	0.0017	0.62
0.0750	40.0410	0.02	0.0020	0.75
0.0875	40.2060	0.02	0.0062	0 87
0.1000	40.1530	0.02	0.0048	1.00
0.1125	40.3020	0.02	0.0086	1.12
0.1250	40.2490	0.02	0.0072	1.24
0.1375	40.2630	0.02	0.0076	1.37
0.1500	40.5300	0.02	0.0143	1 48
0.1625	40.4030	0.02	0.0111	1.61
0.1750	40.4520	0.02	0.0123	1.73
0.1875	40.4460	0.02	0.0122	1.85
0.2000	40.6430	0.02	0.0171	1.97
0.2125	40.5870	0.02	0.0157	2.09
0.2250	40.8960	0.02	0.0234	2.20
0.2375	40.8080	0.02	0.0212	2.33
0.2500	40.9320	0.02	0.0243	2.44

Table E.3: Performance metrics for networks with 39.96 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	63.0000	0.27	1	0.08
0.0250	63.0000	0.10	1	0.16
0.0375	63.0360	0.02	8	0.24
0.0500	63.0260	0.02	4	0.32
0.0625	63.0070	0.02	2	0.40
0.0750	63.0540	0.02	10	0.48
0.0875	63.1010	0.02	13	0.55
0.1000	63.1700	0.02	16	0.63
0.1125	63.1780	0.02	17	0.71
0.1250	63.1990	0.02	20	0.79
0.1375	63.1650	0.02	16	0.87
0.1500	63.1890	0.02	14	0.95
0.1625	63.2550	0.02	23	1.03
0.1750	63.2940	0.02	21	1.11
0.1875	63.3990	0.02	31	1.18
0.2000	63.3680	0.02	20	1.26
0.2125	63.4410	0.02	24	1.34
0.2250	63.6313	0.02	35	1.41
0.2375	63.3788	0.02	24	1.50
0.2500	63.5863	0.02	22	1.57

Table E.4: Performance metrics for networks with 63 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	89.2500	0.43	1	0.06
0.0250	89.2500	0.19	1	0.11
0.0375	89.2600	0.05	3	0.17
0.0500	89.2560	0.02	2	0.22
0 0625	89.2530	0 02	2	0 28
0.0750	89.2790	0.02	7	0.34
0.0875	89.3000	0.02	5	0.39
0.1000	89 4640	0.02	17	0.45
0.1125	89.2760	0.02	4	0.50
0.1250	89.4290	0.02	17	0.56
0.1375	89.2760	0.02	6	0 62
0.1500	89.4050	0.02	13	0.67
0.1625	89.4030	0.02	15	0 73
0.1750	89.3840	0.02	10	0.78
0.1875	89.4230	0.02	15	0.84
0.2000	89.4860	0.02	18	0.89
0.2125	89.5610	0.02	26	0.95
0.2250	89.5790	0.02	27	1.00
0.2375	89.6120	0.02	30	1.06
0.2500	89.5600	0.02	21	1.12

Table E.5: Performance metrics for networks with 89.25 MCPN

Network Stabilisation - Geometric Data

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	7.4680	0.01	5	0.67
0.0250	7.6780	0.01	8	1.30
0.0375	8.0230	0.01	12	1.87
0.0500	8.4560	0.01	13	2.37
0.0625	8.8030	0.01	17	2.84
0.0750	9 4170	0.01	16	3.19
0.0875	9.9750	0.01	15	3.51
0.1000	10.5450	0.01	18	3.79
0.1125	11.0600	0.01	19	4 07
0.1250	11.7930	0.01	16	4.24
0.1375	12.5800	0.01	21	4.37
0.1500	13.6280	0.01	23	4.40
0.1625	14.0870	0.01	22	4.61
0.1750	15.0500	0.01	20	4.65
0.1875	15.6810	0.01	23	4.78
0.2000	16.8050	0.01	22	4.76
0.2125	17.9940	0.01	26	4.72
0.2250	18.8940	0.01	22	4.76
0.2375	19.5910	0.01	25	4.85
0.2500	21.0030	0.01	28	4.76

Table E.6: Performance metrics for networks with 7.49 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	21.1000	0.01	2	0.24
0.0250	21.1550	0.01	7	0.47
0.0375	21.2090	0.01	7	0.71
0.0500	21.3450	0.01	10	0.94
0.0625	21.4830	0.01	10	1.16
0.0750	21.7250	0.01	12	1.38
0.0875	21.8770	0.01	17	1.60
0.1000	21.9230	0.01	15	1.82
0.1125	22.2230	0.01	14	2.03
0.1250	22.2510	0.01	13	2.25
0.1375	22.6870	0.01	17	2.42
0.1500	22.7470	0.01	13	2.64
0.1625	22.9230	0.01	19	2 84
0.1750	23.4140	0.01	17	2.99
0.1875	23.5960	0.01	18	3.18
0.2000	23.9750	0.01	19	3.34
0.2125	24.0350	0.01	22	3.54
0.2250	24.5650	0.01	18	3.66
0.2375	24 8170	0.01	22	3.83
0.2500	25.1780	0.01	23	3.97

Table E.7: Performance metrics for networks with 21.09 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	39.9660	0.01	2	0.13
0.0250	39.9890	0.01	5	0.25
0.0375	39.9880	0.01	5	0.38
0.0500	40.0060	0.01	5	0.50
0.0625	40.0490	0.01	8	0.62
0.0750	40.1470	0.01	10	0.75
0.0875	40.1140	0.01	9	0.87
0.1000	40.1590	0.01	13	1.00
0.1125	40.1870	0.01	11	1.12
0.1250	40.1980	0.01	11	1.24
0.1375	40.2610	0.01	14	1.37
0.1500	40.2920	0.01	12	1.49
0.1625	40.3520	0.01	18	1.61
0.1750	40.3360	0.01	15	1.74
0.1875	40.4490	0.01	21	1.85
0.2000	40.4850	0.01	14	1.98
0.2125	40.5590	0.01	19	2.10
0.2250	40.5610	0.01	15	2.22
0.2375	40.6910	0.01	16	2.33
0.2500	40.6290	0.01	18	2.46

Table E.8: Performance metrics for networks with 39.96 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	63.0090	0.03	3	0.08
0.0250	63.0120	0.01	3	0.16
0.0375	63.0150	0.01	3	0.24
0.0500	63.0450	0.01	6	0.32
0.0625	63.0540	0.01	7	0.40
0.0750	63.1050	0.01	9	0.48
0.0875	63.0930	0.01	8	0.55
0.1000	63.0980	0.01	7	0.63
0.1125	63.1530	0.01	12	0.71
0.1250	63.1630	0.01	15	0.79
0.1375	63.1810	0.01	11	0.87
0.1500	63.2060	0.01	14	0.95
0.1625	63.2610	0.01	14	1.03
0.1750	63.2570	0.01	11	1.11
0.1875	63.2970	0.01	17	1.18
0.2000	63.3200	0.01	15	1.26
0.2125	63.3300	0.01	15	1.34
0.2250	63.3460	0.01	14	1.42
0.2375	63.4070	0.01	14	1.50
0.2500	63.4310	0.01	18	1.58

Table E.9: Performance metrics for networks with 63 MCPN

Loading	MCPN	Attractor Performance	Training Phases	Storage Efficiency
0.0125	89.2510	0.11	1	0.06
0.0250	89.2580	0.02	3	0.11
0.0375	89.2570	0.01	2	0.17
0.0500	89.2730	0.01	4	0.22
0.0625	89.2690	0.01	3	0.28
0.0750	89.3290	0.01	9	0.34
0.0875	89.2960	0.01	8	0.39
0.1000	89.3090	0.01	7	0 45
0.1125	89.3140	0.01	8	0.50
0.1250	89.3080	0.01	6	0.56
0.1375	89.3320	0.01	10	0.62
0.1500	89.3450	0.01	8	0.67
0.1625	89.4180	0.01	16	0.73
0.1750	89.4030	0.01	12	0.78
0.1875	89.4040	0.01	12	0.84
0.2000	89 4290	0.01	14	0.89
0.2125	89.4560	0.01	13	0.95
0.2250	89.4000	0.01	13	1.01
0.2375	89.4470	0.01	12	1.06
0.2500	89.4710	0.01	13	1.12

Table E.10: Performance metrics for networks with 89.25 MCPN

Performance Enhancement – Character Data

Loading \ Connectivity	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
0.0125	-	0.2773	0.7105	0.7468	0.7714	0.7642	0.7730	0.7621	0.7838	0.7899	0.7845	0.7931	0.7842	0.7854	0.7990	0.7869	0.7956	0.7898	0.7911
0.0250	-	0.0122	0.3054	0.6815	0.7096	0.7408	0.7514	0.7444	0.7488	0.7563	0.7753	0.7697	0.7580	0.7714	0.7686	0.7639	0.7778	0.7769	0.7771
0.0375	-	-	0.0207	0.2205	0.6142	0.6479	0.6577	0.6753	0.6646	0.7051	0.7127	0.7387	0.7298	0.7324	0.6882	0.7113	0.7106	0.7219	0.7084
0.0500	-	0.0130	0 0140	0.0357	0.3088	0.6227	0.6240	0.6705	0.6807	0.6964	0.7136	0.6976	0.6893	0.6866	0.7071	0.7160	0.7039	0.7253	0.6963
0.0625	-	-	0.0148	0.0224	0.0243	0.2651	0.3449	0.3738	0.5666	0.5008	0.5798	0.5645	0.5454	0.5802	0.5895	0.5982	0.6176	0.6095	0.5903
0.0750	-	-	0.0150	0.0161	0.0278	0.0748	0.2679	0.4750	0.4782	0.5264	0.5173	0.5863	0.5823	0.5567	0 5771	0.6001	0.6344	0.5807	0.5921
0.0875	-	-	-	0.0156	0.0212	0.0412	0.2296	0.2895	0.3402	0.3581	0.3770	0.4591	0.3772	0.3842	0.4253	0.4002	0.4240	0.4039	0.4670
0.1000	-	-	0.0161	0 0160	0.0201	0.0330	0.1123	0.2687	0.3874	0.5000	0.5823	0.4853	0.4956	0.5386	0.5268	0.5015	0.5259	0.5475	0.5403
0.1125	-	-	-	0.0153	0.0163	0.0162	0.0194	0.1216	0.1506	0.4050	0.1917	0.3076	0.3410	0.5108	0.3683	0.3589	0.4445	0.4423	0.4214
0.1250	-	-	-	0.0178	0.0169	0.0158	0.0164	0.0442	0.1472	0.1130	0.1530	0.1762	0.2440	0.2739	0.3418	0.2557	0 2590	0.2647	0.2685
0.1375	-	-	-	0.0176	-	0.0184	0.0180	0.0177	0.0691	0.1165	0.2134	0.1585	0.2257	0.2875	0.2421	0.2928	0.3063	0.3107	0.3448
0.1500	-	-	-	0.0168	0.0171	0.0169	0.0170	0.0271	0.0410	0.0439	0.1644	0.1072	0.2218	0.1814	0.1899	0.2569	0.2864	0.2443	0.2416
0.1625	-	-	-	-	0.0173	0.0178	0.0180	0.0269	0.0396	0.0778	0.0832	0.3140	0.2124	0.3170	0.2098	0.2061	0.2152	0.3697	0.2541
0.1750	-	-	-	-	-	0.0174	0.0176	0.0180	0.0251	0.0526	0.0551	0.0614	0.1825	0.2154	0.2560	0.1822	0.2465	0.2655	0.3049
0.1875	-	-	-	-	-	0.0186	0.0182	0.0262	0.0181	0.0431	0.0404	0.0695	0.1215	0.1329	0.1575	0.1274	0.1671	0.2276	0.2160
0.2000	-	-	-	-	0.0181	0.0186	0.0183	0.0182	0.0272	0.0264	0.0569	0.0856	0.1557	0.1861	0.1481	0.1785	0.2426	0.2694	0.2522
0.2125	-	-	-	-	-	-	0.0174	-	0.0180	0.0236	0.0399	0.0364	0.0618	0.1033	0.1488	0.1026	0.1236	0.1674	0.2203
0.2250	-	-	-	-	-	-	0.0187	0.0188	0.0192	0.0242	0.0273	0.0407	0.0780	0.0730	0.1554	0.1596	0.1214	0.2374	0.2076
0.2375	-	-	-	-	-	0.0196	0.0187	0.0189	0.0189	0.0187	0.0281	0.0341	0.0739	0.0973	0.1645	0.1718	0.1529	0.1944	0.2285
0.2500	-	-	-	-	-	-	-	0.0194	-	0.0191	0.0193	0.0377	0.0460	0.0483	0.0734	0.0944	0.1278	0.1229	0.2343

Table E.11: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 0.

Loading \ Connectivity	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0 40	0.45	0.50	0 55	0.60	0.65	0.70	0.75	0 80	0 85	0 90	0.95
0.0125	0.0137	0.3630	0.6950	0.7441	0.7239	0.7423	0.7615	0.7582	0.7334	0.7734	0.7647	0.7619	0.7601	0.7541	0.7691	0.7716	0.7686	0.7815	0.7718
0.0250	0.0127	0.0656	0.3471	0.6137	0.6747	0.7159	0.7195	0.7173	0.7318	0.7184	0.7299	0.7453	0.7568	0.7383	0.7468	0.7455	0.7431	0.7482	0.7516
0.0375	0.0131	0.0131	0.0555	0.2289	0.5931	0.6146	0.6669	0.6457	0.6540	0.6935	0.6581	0.6838	0.6790	0.7020	0.6987	0.6694	0.6957	0.6935	0.7097
0.0500	0.0139	0.0138	0.0308	0.1463	0.2067	0.4476	0.5408	0.5438	0.6187	0.6313	0.6514	0.6738	0.6531	0.6570	0.6847	0.7015	0.6701	0.6662	0.6758
0.0625	0.0149	0.0148	0.0148	0.0653	0.1018	0.2304	0.3045	0.3672	0.4037	0.4934	0.5462	0.5247	0.5572	0.5656	0.5429	0.5743	0.5658	0.5777	0.5661
0.0750	0.0160	0.0160	0.0193	0.0219	0.0928	0.1360	0.1999	0.2847	0.3227	0.4558	0.4015	0.4349	0.5088	0.5747	0.5733	0.5653	0.5679	0.6038	0.5834
0.0875	0.0158	0.0160	0.0158	0.0256	0.0405	0.1094	0.1677	0.2302	0.3478	0.3930	0.4029	0.3921	0.3371	0.3997	0.4187	0.3983	0.4317	0.4544	0.3987
0.1000	0.0159	0.0160	0.0162	0.0229	0.0559	0.1284	0.1910	0.2202	0.3040	0.3629	0.4622	0.4881	0.4669	0.5153	0.5064	0.4905	0.5242	0.5310	0.5108
0.1125	0.0164	0.0163	0.0163	0.0162	0.0263	0.0700	0.1291	0.1674	0.1967	0.2205	0.2041	0.2769	0.3193	0.3289	0.3692	0.4091	0.3607	0.3794	0.4069
0.1250	0.0168	0.0167	0.0168	0.0166	0.0300	0.0324	0.1009	0.1284	0.2020	0.1807	0.2302	0.2193	0.2310	0.2278	0.2410	0.2338	0.2352	0.2758	0.2493
0.1375	0.0175	0.0176	0.0177	0.0176	0.0212	0.0246	0.0818	0.1137	0.1316	0.1439	0.1671	0.1835	0.2338	0.1783	0.2231	0.1944	0.2414	0.2498	0.2647
0.1500	0.0196	0.0175	0.0177	0.0176	0.0177	0.0283	0.0344	0.0716	0.1573	0.1347	0.1191	0.1228	0.2687	0.2010	0.2244	0.2143	0.2776	0.3021	0.3033
0.1625	-	0.0178	0.0179	0.0181	0.0179	0.0216	0.0216	0.0621	0.1277	0.1464	0.1416	0.2078	0.2094	0.2644	0.2561	0.2822	0.2798	0.2652	0.3144
0.1750	-	0.0183	0.0181	0.0180	0.0182	0.0179	0.0182	0.0558	0.0962	0.1535	0.1529	0.1808	0.1800	0.1751	0.2387	0.2330	0.1982	0.2210	0.2192
0.1875	-	0.0184	0.0185	0.0183	0.0185	0.0185	0.0220	0.0466	0.0562	0.0757	0.1119	0.1496	0.1311	0.1664	0.1646	0.1824	0.2001	0.1960	0.1842
0.2000	-	0.0185	0.0185	0.0184	0.0185	0.0185	0.0223	0.0428	0.0596	0.0769	0.1193	0.1218	0.1836	0.1829	0.1868	0.1870	0.1907	0.1934	0.2379
0.2125	-	0.0182	0.0184	0.0182	0.0186	0.0183	0.0184	0.0286	0.0567	0.0670	0.1071	0.1063	0.1311	0.1406	0.1726	0.1856	0.1783	0.1672	0.2136
0.2250	-	0.0188	0.0191	0.0188	0.0189	0.0188	0.0187	0.0263	0.0335	0.0509	0.0652	0.1257	0.1179	0.1425	0.1696	0.1668	0.1277	0.2080	0.1876
0.2375	-	0.0193	0.0189	0.0190	0.0191	0.0190	0.0189	0.0224	0.0367	0.0546	0.0758	0.1174	0.1254	0.1354	0.1823	0.1693	0.1976	0.2063	0.2290
0.2500	-	0.0195	0.0192	0.0194	0.0192	0.0191	0.0192	0.0192	0.0229	0.0600	0.0702	0.0981	0.1044	0.1197	0.1152	0.1782	0.1512	0.1776	0.1609

Table E.12: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 1.

Loading \ Connectivity	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0 50	0.55	0 60	0 65	0.70	0 75	0 80	0.85	0 90	0 95
0.0125	-	0.2783	0.5307	0.7126	0.7338	0.7457	0.7748	0.7677	0.7624	0.7667	0.7728	0.7738	0.7760	0.7673	0.7659	0.7677	0.7700	0.7740	0.7727
0.0250	-	0.0350	0.3086	0.5669	0.6508	0.6996	0.7074	0.7048	0.7176	0.7431	0.7455	0.7605	0.7455	0.7501	0.7620	0.7635	0.7446	0.7698	0.7509
0.0375	-	0.0131	0.1015	0.3043	0.4345	0.5936	0.6483	0.6571	0.6687	0.6691	0.6655	0.6863	0.6936	0.6842	0.7080	0.7167	0.6999	0.7036	0.6841
0.0500	-	0.0139	0.0489	0.1228	0.2564	0.4256	0.5451	0.5757	0.6647	0.6530	0.6790	0.7107	0.7017	0.6839	0.7060	0.7060	0.6912	0.6840	0.7002
0.0625	-	0.0147	0.0328	0.1024	0.2115	0.2768	0.3316	0.4243	0.3937	0.4162	0.4706	0.4820	0.4800	0.4670	0.4732	0.5100	0.5033	0.4848	0.5097
0.0750	-	0.0160	0.0192	0.0515	0.1377	0.2202	0.2048	0.3130	0.4163	0.4095	0.4585	0.5165	0.5613	0.5465	0.5488	0.5364	0.5566	0.5877	0.6013
0.0875	-	0.0160	0.0160	0.0368	0.1120	0.1977	0.1654	0.2590	0.2509	0.3318	0.3384	0.3984	0.4265	0.3484	0.3863	0.3690	0.3859	0.4222	0.3811
0.1000	-	0.0160	0.0160	0.0292	0.0707	0.1302	0.2016	0.2393	0.3053	0.3939	0.3735	0.4400	0.4500	0.4713	0.4522	0.5174	0.4660	0.5179	0.4843
0.1125	-	0.0164	0.0163	0.0163	0.0261	0.0682	0.1129	0.1502	0.2140	0.2365	0.2624	0.3166	0.3292	0.2785	0.3328	0.4023	0.3669	0.3677	0.3849
0.1250	-	0.0167	0.0167	0.0168	0.0269	0.0793	0.1200	0.1567	0.2095	0.2255	0.2347	0.2549	0.2530	0.2929	0.2782	0.2614	0.3010	0.2683	0.2862
0.1375	-	0.0178	0.0178	0.0175	0.0281	0.0347	0.1043	0.1248	0.1573	0.1810	0.1674	0.2145	0.2450	0.2476	0.2153	0.2320	0.2996	0.2491	0.2655
0.1500	-	0.0176	0.0175	0.0175	0.0209	0.0385	0.0558	0.1194	0.1436	0.1487	0.2004	0.2069	0.2395	0.2614	0.2925	0.2869	0.2496	0.2888	0.2608
0.1625	-	0.0178	0.0178	0.0180	0.0210	0.0316	0.0457	0.0711	0.0999	0.1261	0.1743	0.1629	0.2279	0.2122	0.2596	0.2947	0.2923	0.2670	0.3004
0.1750	-	0.0180	0.0181	0.0181	0.0181	0.0182	0.0455	0.0792	0.1376	0.1415	0.1270	0.1567	0.1694	0.2159	0.1958	0.2354	0.1953	0.1878	0.1910
0.1875	-	0.0183	0.0186	0.0186	0.0184	0.0185	0.0369	0.0607	0.1315	0.0865	0.1294	0.1612	0.1876	0.1758	0.1620	0.2000	0.1983	0.1872	0.2301
0.2000	-	0.0183	0.0185	0.0185	0.0185	0.0183	0.0295	0.0568	0.0933	0.1313	0.1289	0.1424	0.1438	0.1632	0.2228	0.2339	0.2316	0.2337	0.1975
0.2125	-	0.0183	0.0185	0.0185	0.0186	0.0185	0.0362	0.0541	0.0643	0.1181	0.1017	0.1522	0.1349	0.1474	0.1792	0.1566	0.1942	0.1968	0.2177
0.2250	-	0.0187	0.0189	0.0190	0.0190	0.0188	0.0226	0.0224	0.0787	0.0721	0.0954	0.1466	0.1063	0.1419	0.1983	0.1727	0.1698	0.1719	0.1736
0.2375	-	0.0192	0.0189	0.0189	0.0189	0.0190	0.0191	0.0225	0.0587	0.0762	0.1093	0.1470	0.1150	0.1749	0.1772	0.1627	0.1676	0.1923	0.1870
0.2500	-	0.0192	0.0195	0.0192	0.0194	0.0193	0.0192	0 0191	0.0375	0.0698	0.0878	0.1163	0.1126	0.1184	0.1342	0.1135	0.1440	0.1333	0.1790

Table E.12: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 2.

Loading \ Connectivity	0.05	0.10	0.15	0 20	0.25	0.30	0 35	0.40	0.45	0.50	0.55	0 60	0 65	0 70	0.75	0 80	0 85	0 90	0.95
0.0125	-	0.1202	0.4118	0.6533	0.7106	0.7558	0.7549	0.7737	0.7794	0.7878	0.7905	0.8050	0.8035	0.8003	0.7973	0.7976	0.8015	0.7914	0.7947
0.0250	-	0.0164	0.1497	0.3681	0.5878	0.6245	0.7255	0.7141	0.7360	0.7428	0.7486	0.7423	0.7491	0.7473	0.7555	0.7535	0.7375	0.7394	0.7532
0.0375	-	0.0131	0.0736	0.1777	0.3525	0.5906	0.5957	0.6626	0.6742	0.6911	0.6761	0.7032	0.6773	0.7004	0.7001	0.6981	0.7064	0.7214	0.7147
0.0500	-	0.0139	0.0337	0.1207	0.2407	0.3978	0.5304	0.5478	0.6190	0.6666	0.6916	0.6828	0.6986	0.6945	0.7131	0.7047	0.6857	0.7315	0.6998
0.0625	-	0.0148	0.0299	0.1058	0.1679	0.2566	0.3315	0.3263	0.4212	0.4486	0.5271	0.4946	0.4723	0.4799	0.4796	0.5147	0.5650	0.5834	0.5760
0.0750	-	0.0147	0.0218	0.0312	0.1151	0.2139	0.2865	0.3550	0.4561	0.4590	0.4622	0 5054	0.5179	0.5203	0.5066	0.5176	0.5259	0.5382	0.5617
0.0875	-	-	0.0193	0.0341	0.0993	0.1687	0.2215	0.3107	0.3126	0.3684	0.3735	0.3495	0.3693	0.4003	0.3603	0.4009	0.4326	0.3985	0.3956
0.1000	-	0.0152	0.0161	0.0192	0.0803	0.1365	0.1882	0.2589	0.3029	0.3670	0.4405	0.4371	0.4404	0.4932	0.5249	0.5180	0.4996	0.5156	0.5270
0.1125	-	-	0.0164	0.0195	0.0472	0.0724	0.1222	0.1574	0.2044	0.2104	0.2498	0.2592	0.3017	0.3479	0.3408	0.2978	0.3719	0.3800	0.3551
0.1250	-	-	0.0167	0.0167	0.0395	0.0769	0.1418	0.1597	0.1680	0.2185	0.1887	0.2237	0.2446	0.2821	0.2529	0.2845	0.2736	0.3168	0.2838
0.1375	-	-	0.0177	0.0178	0.0310	0.0554	0.1120	0.1351	0.1674	0.1966	0.2340	0.2024	0.2113	0.2386	0.2633	0.2700	0.2942	0.2885	0.3205
0.1500	-	-	0.0176	0.0214	0.0311	0.0545	0.0624	0.1130	0.1436	0.1689	0.2305	0.2288	0.2370	0.2313	0.2385	0.2724	0.3041	0.3314	0.2989
0.1625	-	-	0.0179	0.0180	0.0178	0.0447	0.0649	0.1057	0.0941	0.1244	0.1876	0.1734	0.2221	0.2133	0.2393	0.2481	0.2424	0.2703	0.3016
0.1750	-	-	0.0180	0.0181	0.0218	0.0220	0.0427	0.0874	0.1109	0.1236	0.1669	0.1688	0.2187	0.1929	0.2040	0.1871	0.1953	0.2320	0.2399
0.1875	-	-	0.0186	0.0186	0.0184	0.0222	0.0501	0.0780	0.1138	0.1105	0.1538	0.1502	0.1765	0.1592	0.2025	0.1861	0.2194	0.1691	0.2241
0.2000	-	-	0.0183	0.0184	0.0185	0.0185	0.0496	0.0493	0.0838	0.1281	0.1283	0.1601	0.1691	0.1887	0.1531	0.1763	0.2349	0.1770	0.2502
0.2125	-	-	0.0185	0.0184	0.0183	0.0185	0.0291	0.0611	0.0936	0.1058	0.1196	0.1341	0.1662	0.1508	0.1587	0.1629	0.1478	0.1884	0.1870
0.2250	-	-	0.0190	0.0191	0.0190	0.0190	0.0262	0.0446	0.0681	0.0985	0.1159	0.0938	0.1433	0.1361	0.1602	0.1449	0.1832	0.1903	0.2109
0.2375	-	-	0.0191	0.0189	0.0188	0.0189	0.0227	0.0302	0.0553	0.0784	0.0997	0.1115	0.1307	0.1743	0.1681	0.1887	0.2220	0.1848	0.2284
0.2500	-	-	0.0194	0.0191	0.0194	0.0194	0.0193	0.0379	0.0450	0.0663	0.0901	0.1012	0.1344	0.1085	0.1451	0.1364	0.1393	0.1686	0.1438

Table E.13: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 3.

Loading \ Connectivity	0 05	0 10	0.15	0.20	0.25	0.30	0.35	0 40	0.45	0 50	0 55	0.60	0 65	0.70	0.75	0.80	0.85	0 90	0 95
0.0125	-	-	-	0.4946	0.6064	0.7118	0.7130	0.7412	0.7379	0.7347	0.7485	0.7483	0.7420	0.7595	0.7483	0.7514	0.7504	0.7547	0.7357
0.0250	-	-	-	0.2024	0.4715	0.6466	0.6483	0.7061	0.7418	0.7438	0.7259	0.7143	0.7456	0.7523	0.7545	0.7411	0.7458	0.7324	0.7491
0.0375	-	-	-	0.1636	0.3335	0.4982	0.6031	0.6817	0.6817	0.6989	0.7185	0.7090	0.7269	0.7126	0.6956	0.7220	0.7022	0.6896	0.7079
0.0500	-	-	-	0.0680	0.2219	0.2778	0.4977	0.5492	0.5568	0.6232	0.6763	0.6448	0.6776	0.6674	0.6707	0.6965	0.6624	0.7089	0.6834
0.0625	-	-	-	0.0442	0.1882	0.2557	0.3607	0.4377	0.4545	0.4595	0.4741	0.5137	0.5126	0.5267	0.5212	0.5284	0.5313	0.5578	0.5818
0.0750	-	-	-	0.0413	0.1459	0.1949	0.2620	0.3537	0.4169	0.4652	0.4791	0.4966	0.5035	0.5465	0.5529	0.5257	0.5628	0.5528	0.5717
0.0875	-	-	-	0.0366	0.0763	0.1080	0.2140	0.2946	0.2906	0.3559	0.3811	0.3524	0.4301	0.3800	0.4191	0.3793	0.4146	0.3987	0.4263
0.1000	-	-	-	0.0226	0.0562	0.0810	0.1927	0.2745	0.3815	0.3871	0.4223	0.4911	0.4965	0.4918	0.4979	0.5556	0.5382	0.5308	0.4994
0.1125	-	-	-	0.0163	0.0331	0.0898	0.1713	0.2080	0.2572	0.2746	0.2223	0.3422	0.3210	0.3163	0.3632	0.4241	0.3890	0.3353	0.4128
0.1250	-	-	-	0.0167	0.0260	0.0775	0.1182	0.1422	0.2152	0.2485	0.1995	0.2095	0.2446	0.2525	0.2360	0.2809	0.2486	0.2793	0.2665
0.1375	-	-	-	0.0175	0.0176	0.0278	0.0926	0.1013	0.1721	0.1831	0.1876	0.2259	0.2415	0.2696	0.2648	0.2810	0.2796	0.2652	0.2907
0.1500	-	-	-	0.0177	0.0207	0.0511	0.0613	0.1006	0.1559	0.1834	0.1947	0.2117	0.2527	0.2467	0.2810	0.2718	0.2954	0.3143	0.2731
0.1625	-	-	-	0.0177	0.0177	0.0315	0.0786	0.0960	0.1748	0.1367	0.1868	0.2507	0.2592	0.2175	0.2985	0.2714	0.2699	0.3269	0.2715
0.1750	-	-	-	0.0181	0.0180	0.0327	0.0492	0.0821	0.1338	0.1450	0.2011	0.1950	0.1859	0.1564	0.1647	0.1642	0.2277	0.1793	0.1885
0.1875	-	-	-	0.0186	0.0184	0.0260	0.0364	0.0637	0.0911	0.1227	0.1127	0.1471	0.1468	0.1631	0.1691	0.2170	0.2251	0.2076	0.2324
0.2000	-	-	-	0.0185	0.0184	0.0220	0.0185	0.0556	0.0798	0.1136	0.1183	0.1385	0.1739	0.1532	0.1855	0.2035	0.2518	0.2344	0.2334
0.2125	-	-	-	0.0184	0.0184	0.0186	0.0219	0.0433	0.0899	0.1276	0.1271	0.1429	0.1462	0.1576	0.1627	0.1577	0.1671	0.1531	0.2173
0.2250	-	-	-	0.0189	0.0189	0.0189	0.0189	0.0481	0.0925	0.0924	0.1279	0.1375	0.1428	0.1238	0.1387	0.1413	0.1592	0.1760	0.1822
0.2375	-	-	-	0.0189	0.0191	0.0190	0.0228	0.0303	0.0647	0.0900	0.1157	0.1309	0.0947	0.1256	0.1408	0.1802	0.1684	0.1937	0.1838
0.2500	-	-	-	0.0191	0.0192	0.0193	0.0194	0.0304	0.0376	0.0872	0.0869	0.1172	0.1319	0.1356	0.1323	0.1634	0.1503	0.1692	0.1892

Table E.14: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 4.

Loading \ Connectivity	0.05	0 10	0.15	0 20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0 60	0.65	0 70	0 75	0 80	0 85	0 90	0 95
0.0125	-	-	-	-	0.5032	0.6305	0.7119	0.7430	0.7507	0.7722	0.7714	0.7619	0.7559	0.7680	0.7633	0.7649	0.7591	0.7722	0.7722
0.0250	-	-	-	-	0.2353	0.5273	0.6338	0.7158	0.7308	0.7505	0.7471	0.7359	0.7434	0.7573	0.7490	0.7596	0.7520	0.7545	0.7487
0.0375	-	-	-	-	0.1498	0.3501	0.5720	0.6350	0.6680	0 6866	0.6847	0.6799	0.6841	0.6899	0.6993	0.7099	0.6972	0.6897	0.7076
0.0500	-	-	-	-	0.0747	0.1884	0.4329	0.5510	0.5443	0.5852	0.6505	0.6556	0.6800	0.6628	0.6863	0.6890	0.6850	0.6958	0.7089
0.0625	-	-	-	-	0.0402	0.1391	0.2873	0.3966	0.4113	0.4575	0.5220	0.5138	0.5196	0.5821	0.4887	0.5741	0.5698	0.5474	0.5593
0.0750	-	-	-	-	0.0409	0.1138	0.2486	0.2966	0.3843	0.4263	0.4851	0.5091	0.5210	0.5318	0.5812	0.5419	0.5885	0.5331	0.5826
0.0875	-	-	-	-	0.0314	0.1008	0.1473	0.2181	0.2707	0.3383	0.3748	0.4294	0.4494	0.4078	0.4055	0.4085	0.4254	0.4496	0.4237
0.1000	-	-	-	-	0.0162	0.0769	0.1242	0.1887	0.2681	0.3235	0.4244	0.4565	0.4733	0.4838	0.4883	0.5383	0.4888	0.5271	0.5261
0.1125	-	-	-	-	0.0194	0.0676	0.1196	0.1701	0.1853	0.2396	0.2734	0.3423	0.3288	0.3445	0.3626	0.3533	0.3610	0.4341	0.4407
0.1250	-	-	-	-	0.0167	0.0456	0.1137	0.1688	0.1707	0.1990	0.2111	0.2133	0.2427	0 2611	0.2436	0.2255	0.2556	0.2569	0.3058
0.1375	-	-	-	-	0.0176	0.0347	0.0926	0.1257	0.1486	0.1821	0.2203	0.2533	0.2534	0.2554	0.2800	0.2911	0.2350	0.3151	0.2987
0.1500	-	-	-	-	0.0209	0.0212	0.0776	0.1134	0.1518	0.1626	0.1669	0.1770	0.2291	0.1865	0.2678	0.2421	0.2536	0.2403	0.2794
0.1625	-	-	-	-	0.0178	0.0179	0.0320	0.0849	0.1535	0.1398	0.1831	0.2520	0.2554	0.2419	0.2774	0.2863	0.2794	0.3022	0.2790
0.1750	-	-	-	-	0.0181	0.0286	0.0318	0.0804	0.1050	0.1125	0.1704	0.1787	0.1710	0.1759	0.2054	0.2108	0.2204	0.2356	0.2199
0.1875	-	-	-	-	0.0185	0.0184	0.0291	0.0324	0.0834	0.1324	0.1274	0.1399	0.1768	0.1802	0.1921	0.2160	0.2306	0.2472	0.2337
0.2000	-	-	-	-	0.0186	0.0182	0.0189	0.0489	0.0876	0.1037	0.1291	0.1635	0.1397	0.2170	0.1533	0.1744	0.2052	0.1936	0.1621
0.2125	-	-	-	-	0.0186	0.0185	0.0256	0.0575	0.0805	0.1005	0.1320	0.1304	0.1573	0.1771	0.1727	0.1689	0.1843	0.1641	0.1522
0.2250	-	-	-	-	0.0188	0.0189	0.0262	0.0369	0.0509	0.0888	0.0953	0.1375	0.1277	0.1341	0.0748	0.1412	0.1157	0.1538	0.1466
0.2375	-	-	-	-	0.0191	0.0188	0.0261	0.0264	0.0616	0.0995	0.0990	0.1536	0.1385	0.1588	0.1906	0.1841	0.1608	0.1629	0.1617
0.2500	-	-	-	-	0.0192	0.0193	0.0190	0.0266	0.0451	0.0762	0.0975	0.1015	0.1131	0.1392	0.1642	0.1350	0.1686	0.1440	0.1145

Table E.15: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 5.

Loading \ Connectivity	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
0.0125	0.0210	0.7555	0.7887	0.7883	0.8038	0.7957	0.8117	0.7942	0.8089	0.8108	0.8020	0.7894	0.8035	0.8157	0.7787	0.7887	0.8003	0.7882	0.7886
0.0250	0.0075	0.3048	0.7269	0.7301	0.7629	0.7719	0.7649	0.7427	0.7874	0.7709	0.7850	0.7824	0.7863	0.7877	0.7926	0.7876	0.7663	0.7880	0.7955
0.0375	0.0081	0.0084	0.4769	0.6808	0.6871	0.7332	0.7439	0.7269	0.7392	0.7318	0.7507	0.7649	0.7560	0.7571	0.7618	0.7728	0.7633	0.7509	0.7560
0.0500	-	0.0088	0.0125	0.3192	0.6271	0.6508	0.6929	0.7074	0.7105	0.7088	0.7128	0.6654	0.6825	0.7229	0.6929	0.7163	0.7117	0.7188	0.7059
0.0625	-	0.0094	0.0093	0.1156	0.5611	0.6315	0.6278	0.6245	0.6588	0.6616	0.6798	0.6840	0.6207	0.6892	0.7189	0.6799	0.6753	0.7029	0.6829
0.0750	-	0.0093	0.0094	0.0203	0.2384	0.5089	0.6110	0.5889	0.5992	0.6408	0.6431	0.5885	0.6316	0.6122	0.6397	0.6413	0.6314	0.6602	0.6427
0.0875	-	0.0093	0.0092	0.0092	0.0579	0.3140	0.4390	0.5427	0.5390	0.5615	0.5581	0.5180	0.5414	0.5855	0.5848	0.5932	0.6009	0.5766	0.5577
0.1000	-	0.0095	0.0096	0.0115	0.0231	0.1486	0.3822	0.4895	0.5061	0.4867	0.5399	0.5421	0.5404	0.5386	0.5248	0.5691	0.5634	0.5021	0.5517
0.1125	-	0.0099	0.0098	0.0098	0.0194	0.0395	0.1736	0.4451	0.4280	0.4495	0.4958	0.4751	0.5128	0.4987	0.5175	0.5189	0.5355	0.5036	0.5016
0.1250	-	-	0.0099	0.0099	0.0119	0.0367	0.1523	0.2825	0.3367	0.4299	0.4945	0.4624	0.4806	0.4852	0.4769	0.4823	0.5103	0.5059	0.4777
0.1375	-	-	0.0096	0.0098	0.0098	0.0157	0.0858	0.2212	0.3038	0.3966	0.4153	0.4104	0.4425	0.4703	0.4462	0.4661	0.4767	0.4435	0.4815
0.1500	-	-	0.0098	0.0099	0.0099	0.0139	0.0311	0.1711	0.1752	0.3510	0.3459	0.3434	0.3707	0.4201	0.3789	0.4552	0.4165	0.4076	0.4547
0.1625	-	-	0.0102	0.0102	0.0102	0.0142	0.0263	0.0540	0.1885	0.2720	0.2953	0.4006	0.3569	0.3889	0.3832	0.3927	0.4054	0.4292	0.4055
0.1750	-	-	0.0100	0.0101	0.0102	0.0102	0.0203	0.0606	0.1014	0.2343	0.2306	0.3266	0.3751	0.3444	0.3792	0.3745	0.3649	0.3764	0.3821
0.1875	-	-	-	0.0103	0.0102	0.0103	0.0123	0.0341	0.1023	0.1857	0.2057	0.2597	0.2996	0.3022	0.3550	0.3291	0.3524	0.3522	0.3537
0.2000	-	-	-	0.0102	0.0102	0.0103	0.0122	0.0281	0.0681	0.1130	0.1709	0.1925	0.2622	0.3275	0.2948	0.3259	0.3400	0.3249	0.3182
0.2125	-	-	-	0.0106	0.0105	0.0104	0.0105	0.0147	0.0210	0.1125	0.1285	0.2082	0.2848	0.2715	0.3072	0.3180	0.3139	0.3364	0.3268
0.2250	-	-	-	0.0103	0.0103	0.0103	0.0103	0.0124	0.0642	0.1112	0.1513	0.1899	0.2054	0.2519	0.3145	0.2584	0.2807	0.3172	0.3210
0.2375	-	-	-	0.0105	0.0105	0.0105	0.0105	0.0165	0.0266	0.0894	0.1479	0.1806	0.1765	0.2582	0.2636	0.2553	0.3172	0.2964	0.3017
0.2500	-	-	-	0.0104	0.0105	0.0105	0.0105	0.0125	0.0208	0.0428	0.1303	0.1730	0.1978	0.2108	0.2258	0.2725	0.2587	0.2902	0.2987

Table E.16: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 0.

Loading \ Connectivity	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
0.0125	0.0443	0.6172	0.7659	0.7583	0.7606	0.7612	0.7735	0.7728	0.7750	0.7782	0.7680	0.7659	0.7856	0.7825	0.7800	0.7760	0.7814	0.7725	0.7784
0.0250	0.0078	0.1634	0.5539	0.7124	0.7233	0.7293	0.7505	0.7429	0.7341	0.7641	0.7599	0.7710	0.7666	0.7507	0.7594	0.7519	0.7651	0.7642	0.7667
0.0375	0.0084	0.0509	0.2221	0.4590	0.6065	0.6831	0.6907	0.6917	0.7115	0.7098	0.7397	0.7443	0.7384	0.7355	0.7515	0.7253	0.7381	0.7082	0.7329
0.0500	0.0088	0.0158	0.0804	0.2786	0.4527	0.5982	0.5932	0.6210	0.6200	0.6487	0.6538	0.6884	0.6757	0.6840	0.6695	0.6667	0.6776	0.6880	0.6909
0.0625	0.0093	0.0094	0.0589	0.1306	0.2435	0.4260	0.5184	0.5682	0.6153	0.6319	0.6080	0.6243	0.6366	0.6501	0.6681	0.6353	0.6717	0.6796	0.6698
0.0750	0.0094	0.0094	0.0243	0.1014	0.2477	0.3027	0.4326	0.5100	0.4990	0.5594	0.5310	0.5490	0.6059	0.5975	0.5885	0.5238	0.5813	0.6003	0.5784
0.0875	0.0092	0.0092	0.0149	0.0347	0.1038	0.1710	0.3406	0.4478	0.3814	0.4598	0.4755	0.5170	0.5292	0.5157	0.5262	0.5844	0.5722	0.5792	0.5429
0.1000	0.0096	0.0096	0.0115	0.0341	0.0913	0.1838	0.2654	0.3410	0.4078	0.4583	0.4733	0.4829	0.4716	0.4928	0.5308	0.5432	0.5324	0.5309	0.5228
0.1125	0.0098	0.0098	0.0098	0.0175	0.0816	0.1174	0.2417	0.2711	0.3770	0.4270	0.3997	0.4185	0.4748	0.4547	0.4428	0.4851	0.4759	0.5023	0.5175
0.1250	0.0098	0.0099	0.0099	0.0158	0.0397	0.1179	0.1813	0.2002	0.2802	0.3203	0.3688	0.3940	0.4150	0.4101	0.4553	0.4686	0.4531	0.5109	0.4865
0.1375	0.0098	0.0098	0.0098	0.0156	0.0482	0.0759	0.1483	0.1753	0.2235	0.2654	0.3044	0.3414	0.3526	0.3605	0.3774	0.3601	0.4115	0.3862	0.4021
0.1500	0.0097	0.0099	0.0099	0.0099	0.0195	0.0839	0.1162	0.1645	0.2067	0.2603	0.2319	0.3142	0.3149	0.3283	0.3990	0.3872	0.3695	0.4106	0.4230
0.1625	0.0102	0.0102	0.0102	0.0122	0.0162	0.0340	0.0976	0.1487	0.1735	0.1901	0.2711	0.2439	0.3011	0.3096	0.3364	0.3466	0.3824	0.3448	0.3694
0.1750	0.0102	0.0102	0.0102	0.0102	0.0202	0.0460	0.0975	0.1337	0.1234	0.1846	0.2244	0.2303	0.2324	0.3390	0.3231	0.3247	0.3139	0.3176	0.3531
0.1875	-	0.0102	0.0102	0.0102	0.0102	0.0183	0.0514	0.0899	0.1226	0.1750	0.1898	0.2357	0.2642	0.2764	0.2964	0.2695	0.3031	0.3390	0.3125
0.2000	-	0.0102	0.0102	0.0102	0.0102	0.0102	0.0183	0.0787	0.1008	0.1396	0.1762	0.1931	0.2566	0.2406	0.2884	0.2875	0.3012	0.3038	0.3332
0.2125	-	0.0105	0.0105	0.0105	0.0105	0.0168	0.0513	0.0694	0.0642	0.1505	0.1243	0.2307	0.2226	0.2445	0.2417	0.2822	0.2952	0.2903	0.2950
0.2250	-	0.0103	0.0103	0.0103	0.0103	0.0145	0.0185	0.0601	0.0930	0.1040	0.1356	0.1364	0.1694	0.2152	0.2098	0.2444	0.2763	0.2912	0.3022
0.2375	-	0.0105	0.0105	0.0105	0.0105	0.0105	0.0186	0.0479	0.0917	0.1025	0.1328	0.1270	0.1856	0.1767	0.2295	0.2669	0.2508	0.2715	0.2840
0.2500	-	0.0104	0.0105	0.0105	0.0105	0.0105	0.0125	0.0349	0.0507	0.1039	0.1164	0.1402	0.1675	0.1706	0.2004	0.2089	0.1984	0.2484	0.2635

Table E.17: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 1.

Loading \ Connectivity	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
0.0125	-	0.3243	0.7058	0.7449	0.7600	0.7872	0.7315	0.7446	0.7646	0.7706	0.8069	0.7874	0.7875	0.7891	0.7837	0.7683	0.8165	0.7984	0.7503
0.0250	-	0.0798	0.2531	0.5536	0.6398	0.7222	0.7716	0.7366	0.7528	0.7654	0.7612	0.7663	0.7609	0.7621	0.7830	0.7465	0.7787	0.7895	0.7813
0.0375	-	0.0325	0.2135	0.2746	0.4302	0.4193	0.5801	0.5854	0.7097	0.7547	0.6957	0.7429	0.7421	0.7578	0.7462	0.7424	0.7698	0.7284	0.7120
0.0500	-	0.0085	0.1300	0.2340	0.2892	0.3988	0.4815	0.4939	0.5591	0.6429	0.6003	0.5329	0.6329	0.5681	0.7037	0.6258	0.6777	0.7057	0.6872
0.0625	-	0.0092	0.0976	0.1413	0.1231	0.3201	0.2122	0.3842	0.4238	0.4670	0.5199	0.4916	0.5089	0.5557	0.5757	0.5924	0.5160	0.5960	0.5354
0.0750	-	0.0091	0.0181	0.0799	0.1132	0.2898	0.4281	0.4445	0.4100	0.4936	0.4192	0.4636	0.5537	0.6028	0.5382	0.6337	0.6176	0.6383	0.6344
0.0875	-	0.0089	0.0266	0.0526	0.1436	0.1665	0.1899	0.4530	0.4170	0.4303	0.5170	0.4718	0.4762	0.5411	0.5218	0.5508	0.5408	0.6125	0.6021
0.1000	-	0.0101	0.0204	0.0689	0.1173	0.1524	0.2511	0.2591	0.3050	0.3351	0.3780	0.4670	0.4743	0.4695	0.5538	0.4729	0.5936	0.5153	0.5444
0.1125	-	0.0099	0.0099	0.0482	0.0848	0.1805	0.0577	0.2296	0.2303	0.3590	0.2362	0.4080	0.4007	0.3329	0.3980	0.4938	0.4408	0.4176	0.4462
0.1250	-	0.0102	0.0102	0.0103	0.0304	0.1353	0.1785	0.2280	0.3196	0.3508	0.2976	0.3576	0.3829	0.3565	0.4833	0.4746	0.4259	0.4111	0.5103
0.1375	-	0.0096	0.0096	0.0285	0.0566	0.1614	0.1701	0.1014	0.2657	0.2721	0.3440	0.3219	0.3369	0.4000	0.4055	0.4313	0.4376	0.4350	0.4438
0.1500	-	0.0097	0.0097	0.0098	0.0290	0.0940	0.1374	0.1372	0.1365	0.2509	0.2206	0.2889	0.3475	0.4327	0.4316	0.4520	0.4273	0.4382	0.4333
0.1625	-	0.0099	0.0101	0.0101	0.0200	0.0598	0.1320	0.1585	0.1406	0.2024	0.2869	0.3092	0.2869	0.3167	0.3012	0.3254	0.2874	0.3316	0.3810
0.1750	-	0.0103	0.0103	0.0102	0.0303	0.0302	0.1341	0.0973	0.0595	0.1710	0.1431	0.2818	0.2594	0.2653	0.3697	0.3561	0.2982	0.3505	0.2451
0.1875	-	0.0101	0.0101	0.0101	0.0199	0.0781	0.1157	0.1239	0.1243	0.1239	0.1587	0.1927	0.2418	0.2363	0.2571	0.2338	0.3391	0.2110	0.3256
0.2000	-	0.0100	0.0099	0.0099	0.0197	0.0397	0.0759	0.1319	0.1743	0.1494	0.1653	0.2241	0.1760	0.2484	0.2567	0.2784	0.2927	0.2642	0.3449
0.2125	-	0.0103	0.0104	0.0103	0.0207	0.0210	0.0408	0.0903	0.1269	0.1085	0.1369	0.1364	0.2653	0.2467	0.1998	0.3039	0.2866	0.2872	0.2961
0.2250	-	0.0103	0.0102	0.0101	0.0204	0.0305	0.0883	0.0881	0.1079	0.1158	0.1439	0.1790	0.2122	0.1608	0.2348	0.2744	0.2516	0.3245	0.2909
0.2375	-	0.0102	0.0103	0.0103	0.0103	0.0204	0.0603	0.0305	0.0694	0.0691	0.1075	0.1606	0.1955	0.2367	0.2366	0.1779	0.2285	0.2920	0.2362
0.2500	-	0.0103	0.0102	0.0103	0.0102	0.0204	0.0202	0.0407	0.1352	0.1344	0.1535	0.1171	0.1696	0.2129	0.2124	0.2127	0.3216	0.2920	0.2537

Table E.18: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 2.

Loading \ Connectivity	0.05	0.10	0.15	0.20	0 25	0.30	0.35	0.40	0 45	0.50	0 55	0.60	0.65	0 70	0.75	0 80	0.85	0.90	0 95
0.0125	-	0.0188	0.3014	0.5734	0.7360	0.7733	0.7695	0.7801	0.7746	0.7805	0.7812	0.7902	0.7917	0.7764	0.7798	0.7870	0.7823	0.7857	0.7724
0.0250	-	0.0078	0.1181	0.3700	0.4809	0.5483	0.6532	0.6467	0.7244	0.7492	0.7658	0.7604	0.7688	0.7762	0.7595	0.7770	0.7471	0.7572	0.7745
0.0375	-	0.0082	0.0657	0.1954	0.3429	0.4662	0.5745	0.5991	0.6474	0.6574	0.6954	0.7147	0.6917	0.7208	0.7108	0.7225	0.7103	0.7368	0.7226
0.0500	-	0.0089	0.0326	0.1052	0.2822	0.3748	0.4270	0.5030	0.5651	0.5985	0.6331	0.6296	0.6530	0.6494	0.6593	0.6574	0.6650	0.6841	0.6805
0.0625	-	0.0100	0.0112	0.0978	0.1750	0.3447	0.3463	0.4679	0.4805	0.5400	0.5632	0.5925	0.6436	0.6096	0.6207	0.6287	0.6492	0.6457	0.6513
0.0750	-	-	0.0150	0.0560	0.1743	0.2331	0.2885	0.3922	0.4192	0.4655	0.4643	0.5171	0.4957	0.5648	0.5853	0.5507	0.5550	0.5641	0.5730
0.0875	-	-	0.0111	0.0452	0.1195	0.1798	0.2481	0.3145	0.3734	0.4006	0.4307	0.4335	0.4975	0.5227	0.5072	0.5422	0.5495	0.5376	0.5930
0.1000	-	-	0.0096	0.0339	0.0878	0.1701	0.2214	0.2264	0.3299	0.3558	0.4004	0.4077	0.4561	0.4611	0.4546	0.4841	0.5026	0.5225	0.5116
0.1125	-	-	0.0098	0.0213	0.0704	0.1422	0.2068	0.2275	0.3048	0.3120	0.3638	0.4022	0.4010	0.4573	0.4259	0.4487	0.5000	0.4663	0.4955
0.1250	-	-	0.0099	0.0236	0.0363	0.0681	0.1821	0.2390	0.2962	0.3018	0.2772	0.3978	0.4032	0.3941	0.3914	0.4261	0.4336	0.3775	0.4465
0.1375	-	-	0.0098	0.0098	0.0626	0.1225	0.1733	0.1954	0.2335	0.2764	0.3257	0.2814	0.3718	0.3660	0.3632	0.4283	0.3753	0.4376	0.4552
0.1500	-	-	0.0099	0.0119	0.0256	0.0577	0.1118	0.1505	0.1813	0.2107	0.2608	0.2908	0.3134	0.3094	0.3761	0.3982	0.4193	0.4069	0.4307
0.1625	-	-	0.0102	0.0142	0.0340	0.0856	0.1104	0.1804	0.1663	0.2347	0.2592	0.2912	0.2676	0.3229	0.3010	0.3271	0.3465	0.3787	0.3643
0.1750	-	-	0.0101	0.0102	0.0182	0.0457	0.0898	0.1289	0.1697	0.1835	0.2155	0.2476	0.2552	0.3055	0.3165	0.3229	0.3364	0.3724	0.3542
0.1875	-	-	0.0102	0.0102	0.0162	0.0224	0.0594	0.1052	0.1285	0.1499	0.2054	0.2131	0.2598	0.2393	0.2769	0.2814	0.2923	0.3374	0.3208
0.2000	-	-	0.0103	0.0102	0.0123	0.0461	0.0561	0.0879	0.1406	0.1682	0.1736	0.2149	0.2118	0.2140	0.2733	0.2900	0.2704	0.2793	0.3289
0.2125	-	-	0.0105	0.0105	0.0188	0.0512	0.0634	0.0711	0.1187	0.1660	0.1540	0.2084	0.2034	0.2072	0.2369	0.2903	0.2851	0.3011	0.3050
0.2250	-	-	0.0104	0.0103	0.0124	0.0267	0.0426	0.0950	0.0911	0.1318	0.1613	0.1725	0.1940	0.2084	0.2403	0.2539	0.2412	0.2367	0.2459
0.2375	-	-	0.0105	0.0105	0.0147	0.0229	0.0349	0.0673	0.1016	0.1256	0.1394	0.1802	0.1528	0.2336	0.2236	0.2304	0.2419	0.2914	0.2443
0.2500	-	-	0.0105	0.0104	0.0105	0.0146	0.0390	0.0432	0.0799	0.1261	0.1259	0.1410	0.1840	0.1918	0.2109	0.2030	0.2440	0.1968	0.2561

Table E.19: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 3.

Loading \ Connectivity	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0 40	0 45	0.50	0 55	0.60	0.65	0.70	0.75	0.80	0 85	0.90	0 95
0.0125	-	-	-	0.3476	0.5620	0.6858	0.7612	0.7738	0.7642	0.7862	0.7921	0.7893	0.7817	0.7794	0.7717	0.7603	0 7682	0.7777	0.7714
0.0250	-	-	-	0.1732	0.3628	0.5170	0.6357	0.6643	0.6933	0.7249	0.7572	0.7690	0.7668	0.7750	0.7726	0.7553	0.7621	0.7542	0.7655
0.0375	-	-	-	0.0794	0.2837	0.3985	0.4601	0.4992	0.5905	0.6194	0.6580	0.6669	0.6914	0.7128	0.7153	0.7157	0.7314	0.7327	0.7482
0.0500	-	-	-	0.0419	0.1905	0.2879	0.3946	0.4669	0.4835	0.5590	0.6214	0.6055	0.6457	0.6354	0.6679	0.6733	0.6776	0.6854	0.6834
0.0625	-	-	-	0.0246	0.1273	0.2028	0.3121	0.4055	0.4464	0.5065	0.5640	0.5757	0.5902	0.6010	0.6239	0.6153	0.6333	0.6611	0 6721
0.0750	-	-	-	0.0257	0.1041	0.2309	0.3081	0.3658	0.3943	0.4560	0.4205	0.5228	0.5192	0.5098	0.5291	0.5639	0.5709	0.5810	0.6074
0.0875	-	-	-	0.0185	0.0721	0.1600	0.2206	0.2866	0.3329	0.3999	0.4725	0.4601	0.4930	0.4839	0.5060	0.4898	0.5161	0.5147	0.5448
0.1000	-	-	-	0.0115	0.0432	0.1101	0.1640	0.2393	0.2889	0.3472	0.3388	0.4163	0.4623	0.4735	0.5010	0.4903	0.5377	0.5201	0.5195
0.1125	-	-	-	0.0136	0.0272	0.1096	0.1611	0.2186	0.2472	0.2943	0.3090	0.3752	0.4270	0.4108	0.4014	0.4607	0.4691	0.4944	0.4732
0.1250	-	-	-	0.0099	0.0328	0.0617	0.1375	0.2196	0.2338	0.3001	0.3265	0.3364	0.3954	0.4162	0.3985	0.4196	0.4255	0.4805	0.4600
0.1375	-	-	-	0.0098	0.0233	0.0905	0.0975	0.1964	0.2198	0.2238	0.2966	0.2600	0.3186	0.3300	0.3880	0.3822	0.4019	0.4119	0.4039
0.1500	-	-	-	0.0118	0.0179	0.0354	0.0994	0.1444	0.2010	0.2218	0.2675	0.2904	0.2997	0.3257	0.3828	0.3832	0.3893	0.3929	0.4305
0.1625	-	-	-	0.0102	0.0204	0.0438	0.1069	0.1428	0.1729	0.1840	0.2469	0.2724	0.2820	0.3057	0.3047	0.3242	0.3516	0.3350	0.3850
0.1750	-	-	-	0.0102	0.0101	0.0420	0.0802	0.1201	0.1476	0.1900	0.2219	0.2377	0.2995	0.2937	0.2995	0.3278	0.3309	0.3223	0.3621
0.1875	-	-	-	0.0102	0.0123	0.0264	0.0579	0.0995	0.1137	0.1258	0.1833	0.1962	0.2012	0.2700	0.2524	0.2480	0.3354	0.2938	0.3433
0.2000	-	-	-	0.0102	0.0102	0.0323	0.0499	0.0743	0.1304	0.1569	0.1772	0.1323	0.2044	0.2169	0.2115	0.2760	0.2651	0.2518	0.2600
0.2125	-	-	-	0.0105	0.0127	0.0188	0.0453	0.0949	0.1092	0.1374	0.1506	0.1835	0.2245	0.2378	0.1961	0.2106	0.2881	0.2824	0.2794
0.2250	-	-	-	0.0103	0.0103	0.0165	0.0403	0.0603	0.0932	0.1004	0.1135	0.1543	0.1713	0.2335	0.2088	0.2374	0.2246	0.2634	0.2773
0.2375	-	-	-	0.0105	0.0105	0.0188	0.0310	0.0650	0.0938	0.1130	0.1388	0.1497	0.1543	0.2081	0.1929	0.2371	0.2678	0.2233	0.2518
0.2500	-	-	-	0.0105	0.0105	0.0168	0.0208	0.0432	0.0710	0.1114	0.1188	0.1402	0.1327	0.1145	0.1786	0.2074	0.2246	0.2350	0.2753

Table E.20: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 4.

Loading \ Connectivity	0.05	0 10	0.15	0.20	0 25	0.30	0.35	0.40	0.45	0.50	0 55	0 60	0 65	0 70	0 75	0 80	0 85	0 90	0 95
0.0125	-	-	-	-	0.4519	0.5328	0.6579	0.7403	0.7624	0.7641	0.7777	0.7992	0.7723	0.7821	0.7856	0.7175	0.7497	0.7277	0.7637
0.0250	-	-	-	-	0.1166	0.4658	0.4993	0.6240	0.6840	0 6591	0.7337	0.7415	0.7463	0.7404	0.7490	0.7460	0.8097	0.7835	0.7328
0.0375	-	-	-	-	0.0702	0.0955	0.3467	0.5474	0.5912	0.6363	0.6561	0.6869	0.6527	0.7320	0.7714	0.6713	0.6870	0.7222	0.7319
0.0500	-	-	-	-	0.1392	0.1935	0.3318	0.3671	0.3971	0.5420	0.6885	0.6289	0.6801	0.6318	0.6974	0.7437	0.6889	0.6745	0.7585
0.0625	-	-	-	-	0.0363	0.2029	0.2763	0.2892	0.4484	0.5645	0.5809	0.5547	0.6183	0.6266	0.6122	0.6478	0.6544	0.6510	0.6261
0.0750	-	-	-	-	0.0098	0.1042	0.2053	0.2586	0.3725	0.3973	0.3872	0.3800	0.4069	0.4257	0.4080	0.4652	0.4640	0.3876	0.5350
0.0875	-	-	-	-	0.0094	0.1188	0.1758	0.1681	0.3217	0.3141	0.3461	0.3745	0.5136	0.5221	0.5388	0.5208	0.5156	0.5264	0.5912
0.1000	-	-	-	-	0.0182	0.1296	0.1385	0.3097	0.2466	0.3446	0.3631	0.3586	0.3509	0.5158	0.3958	0.5154	0.4567	0.4636	0.4599
0.1125	-	-	-	-	0.0304	0.1438	0.1869	0.2121	0.2910	0.3239	0.3599	0.4482	0.4044	0.3363	0.5034	0.4829	0.4252	0.5452	0.4715
0.1250	-	-	-	-	0.0094	0.0373	0.1343	0.1432	0.2224	0.2074	0.3490	0.3202	0.3339	0.3553	0.4084	0.4761	0.4652	0.3955	0.4454
0.1375	-	-	-	-	0.0202	0.0302	0.1260	0.1161	0.2840	0.2212	0.3053	0.3857	0.3583	0.3408	0.4516	0.4201	0.4119	0.4131	0.4122
0.1500	-	-	-	-	0.0103	0.0303	0.0503	0.1254	0.1860	0.2430	0.2514	0.3128	0.2838	0.3337	0.3256	0.3589	0.3354	0.2852	0.2360
0.1625	-	-	-	-	0.0100	0.0101	0.0493	0.0867	0.1926	0.2425	0.2174	0.3161	0.3187	0.3505	0.3916	0.2471	0.3553	0.3959	0.3468
0.1750	-	-	-	-	0.0102	0.0202	0.0102	0.0803	0.1970	0.1793	0.1612	0.1970	0.2135	0.2860	0.3079	0.3781	0.2543	0.4196	0.2993
0.1875	-	-	-	-	0.0201	0.0102	0.0201	0.1161	0.1336	0.1155	0.1506	0.1701	0.1681	0.2085	0.1496	0.1772	0.2421	0.2826	0.3357
0.2000	-	-	-	-	0.0104	0.0103	0.0103	0.1081	0.1254	0.1620	0.1695	0.1977	0.2226	0.1614	0.2309	0.2935	0.2303	0.3214	0.2223
0.2125	-	-	-	-	0.0107	0.0108	0.0107	0.0828	0.0728	0.1025	0.0924	0.1215	0.1857	0.2192	0.1770	0.2795	0.2375	0.2776	0.3225
0.2250	-	-	-	-	0.0105	0.0104	0.0208	0.0411	0.0997	0.0803	0.1364	0.1987	0.1816	0.2481	0.3118	0.2314	0.2644	0.3551	0.3341
0.2375	-	-	-	-	0.0104	0.0104	0.0306	0.0205	0.0700	0.1091	0.1721	0.1892	0.1561	0.2791	0.1636	0.1809	0.2711	0.1183	0.2457
0.2500	-	-	-	-	0.0105	0.0105	0.0104	0.0413	0.0104	0.1000	0.1644	0.1101	0.2338	0.1819	0.1550	0.2251	0.2399	0.2578	0.2174

Table E.21: Attractor performance by connectivity and loading level for networks with initial neighbourhoods of size 5.

An experimental assessment of the performance of several associative memory models.

S. P. Turvey, S.P.Hunt, N.Davey, R.J.Frank [1]

Abstract

The performance characteristics of four different associative memory models are examined. The models differ in the training algorithm employed, although all four employ algorithms that are iterative, and use local information. They are classified using the method of Abbott [1], their attractor performance is examined, and the time taken to train them is measured.

1. Introduction

The dynamics and performance of the Hopfield associative memory model have been thoroughly investigated and are well understood. Several alternative training algorithms have been proposed, each of which leads to an increase in capacity over the original Hopfield model, and an improvement in attractor performance, usually at the expense of an increase in training time.

This paper compares the performance of a number of such high capacity models, with respect to training time, attractor performance and the stability of stored patterns. The work has two goals. First, to classify the models in question using the method developed by Abbott [1]. Second, to evaluate the models against a consistent set of criteria in order to ascertain which of them gives the best balance of performance.

2. Models examined

2.1 Common properties

Each model employs a fully connected network of N bipolar (+1/-1) processing elements, as used in the Hopfield model. Each network is trained using a set, Π, of N-ary, bipolar pattern vectors, $\{\xi^p\}$. The N by N weight matrix which results will be denoted by $\mathbf{W}$, and the state (output) of the i'th unit by S_i.

The *local field* (input) of the i'th unit, h_i, is given by:

$$h_i = \sum_{j \neq i} w_{ij} S_j$$

The *aligned local field* of the i'th unit for pattern ξ^p is

$$h_i \xi_i^p$$

If all aligned local fields for a ξ^p are non-negative it is guaranteed to be stable.

The temporal evolution of unit states during recall is governed by:

$$S_i' = \begin{cases} 1 & \text{if } h_i > 0 \\ -1 & \text{if } h_i < 0 \\ S_i & \text{if } h_i = 0 \end{cases}$$

Unit states may be updated either synchronously or asynchronously. All models investigated here employ asynchronous, random-order updates, and updating continues until the network reaches a stable state. These dynamics, coupled with a symmetric weight matrix, guarantee simple point attractors [2].

Each ξ^p that is a stable state of the trained network is known as a *fundamental memory*. The *capacity* of a network, C, is the maximum number of fundamental memories it can hold. The *loading* of a network, α, is a measure of the size of the training set relative to the number of processing elements in the network, giving

$$\alpha = \frac{\#(\Pi)}{N} \qquad \text{and} \qquad \alpha_{max} = \frac{C}{N}$$

2.2 The Iterative Local Learning rule (ILL)

This learning rule, devised by Diederich and Opper [3], is similar to the perceptron convergence procedure. The algorithm attempts to push the values of all units' aligned local fields to be greater than or equal to the training threshold, T, for all ξ^p. Algorithmically, this rule is as follows:

Beginning with a zero weight matrix
Repeat until all aligned local fields are correct
For each training pattern, ξ^p, in turn
Clamp the pattern onto the network
For each processing element in turn
If $h_i^p \xi_i^p < T$, change the weights on connections into unit i according to:

$$\Delta w_{ij} = \frac{\xi_i^p \xi_j^p}{N} \quad (i \neq j)$$

Note that the resulting $\mathbf{W}$ will have a zero diagonal, but is unlikely to be symmetric. A variant of this rule exists that enforces $\Delta w_{ij} = \Delta w_{ji}$ for each weight change, thus guaranteeing symmetry and, hence, simple point

[1] Department of Computer Science, University of Hertfordshire, Hatfield, Herts, AL10 9AB. United Kingdom
e-mail: {s.p.turvey, s.p.hunt, n.davey, r.j.frank}@herts.ac.uk

attractors. We have chosen not to examine it here because its attractor performance is not markedly superior to the ILL rule (see [4] for details).

2.3 The Iterative Local Learning with Equal Fields rule (ILL-Eq)

Diederich and Opper [3] proposed a modification to ILL in which weights are changed so that the aligned local fields of all units asymptotically approach 1 for every pattern. In the implementation employed here, training continues until the value of every aligned local field falls within the range 0.998 .. 1.002.

Training proceeds as follows:

Beginning with a zero weight matrix
Repeat until all aligned local fields are correct
For each training pattern, ξ^p, in turn
Clamp the pattern onto the network
For each processing element in turn
Update incoming weights according to:

$$\Delta w_{ij} = \frac{\left(1 - h_i^p S_i^p\right) S_i^p S_j^p}{N} \quad (i \neq j)$$

Performance may be varied by changing the acceptable range of values for aligned local fields. The effect this has on attractor performance and training time is the subject of work to be published.

2.4 The Krauth - Mezard learning rule (KM)

Another modification to ILL, this rule was proposed by Krauth and Mezard [5]. It attempts to present each training pattern an optimal number of times. At each unit, the pattern with the smallest aligned local field is chosen for presentation. Once again, weights are changed until all aligned local fields are greater than or equal to the training threshold, T:

Beginning with a zero weight matrix
Repeat until all aligned local fields are correct
For each unit, i, in turn
Select the pattern, ξ^p, with the smallest aligned local field for this unit
Update the incoming weights according to:

$$\Delta w_{ij} = \frac{\xi_i^p \xi_j^p}{N} \quad (i \neq j)$$

This rule has been shown produce optimal γ values, γ is described in section 3.1.

2.5 The Blatt - Vergini learning rule (BV)

Blatt and Vergini [6] propose a training algorithm that is guaranteed to find an appropriate weight matrix within a finite number of presentations of each pattern.

The minimum number of presentations to perform, P, is calculated as being the smallest integer conforming to:

$$P \geq \log_k \left(\frac{N}{(1-T)^2} \right)$$

where k and T are real valued constants such that $1 < k \leq 4$ and $0 \leq T < 1$, and N is the number of units in the network. k is referred to as the *memory coefficient* of the network, because the larger it is, the fewer steps are required to train the network. In this work, k=4 and T=0.5 for all networks trained by this rule.

The algorithm is as follows:

Beginning with a zero weight matrix
For each pattern in turn
Clamp the pattern onto the network
For m := 1 to P
For each processing element in turn
Update incoming weights according to:

$$\Delta w_{ij} = \left(\frac{k^{m-1}}{N} \right) \left(\xi_i^p - h_i \right) \left(\xi_j^p - h_j \right)$$

Remove all self-connections

Note that patterns are added incrementally without interfering with patterns learnt previously.

2.6 Relationship to the pseudo-inverse rule

The algorithms employed in the ILL-Eq and BV models are both designed to generate weight matrices that are approximations of the weight matrix generated by the *pseudo-inverse* rule of Personnaz et al [7]. According to this rule $W=\Xi\Xi^I$, where Ξ is the matrix whose columns are the ξ^p, and Ξ^I is its pseudo-inverse.

ILL-Eq and BV both employ iterative learning algorithms that use local information to generate a weight matrix, $W \approx \Xi\Xi^I$, with its diagonal set to zero.

Whilst the BV rule guarantees a solution to the problem within a finite, and calculable, number of iterations through the training set, there is no upper bound on the number of iterations that may be required for the ILL-Eq to satisfy its stopping criterion. Blatt and Vergini also state there to be no restrictions on the training set with respect to correlation or linear dependency.

Interestingly, Gorodnichy [8] has demonstrated that optimum performance in pseudo-inverse trained networks is obtained when the weight matrix has a non-zero diagonal (specifically, a scaled-down version of the diagonal generated by the pseudo-inverse rule). We considered modifying the BV rule to take this into account, but chose not to do so since we have found the improvements to be relatively small.

3. Experimental procedure

3.1 γ distribution analysis

Abbott [1] identified three classes of associative memory, characterised by the distribution of γ values for a trained network. The γ value for unit i for the pattern, ξ^p, is obtained by dividing the aligned local field by the magnitude of the incoming weight vector:

$$\gamma_i^p = \frac{h_i \xi_i^p}{|W_i|}$$

For each network we may obtain a set, Γ, containing the γ values for all the network's fundamental memories.

Abbott's classification system is based upon the distribution of γ values in Γ. The three classes are:

1) Networks with a Gaussian distribution of γ values
2) Networks with all γ values the same ($\forall\gamma_i \bullet \gamma_i = \gamma_0$)
3) Networks with a clipped Gaussian distribution of γ values, where $\forall\gamma_i \bullet \gamma_i \geq \gamma_{min}$

Class 1 includes networks trained using Hopfield's algorithm, so they are referred to as Hopfield-type networks. Interestingly, Abbott calculates that the upper bound for α_{max} in Class 1 is 1.4, which is much higher than α_{max} for the 'vanilla' Hopfield network.

Class 2 is made up of networks trained using the *pseudo-inverse* rule, or derivatives thereof. The capacity of networks trained using this rule is N linearly independent patterns (giving $\alpha_{max} \approx 1$).

Class 3 is known as the Gardner class, after the work of Gardner [9], whose training algorithm gives networks with $\alpha_{max}=2$.

3.2 Attractor performance

Kanter and Sompolinsky's [10] R value is used as a measure of attractor performance:

$$R = \left\langle \left\langle \left[\frac{1-m_o}{1-m_1} \right] \right\rangle \right\rangle$$

A series of sample starting states are chosen, each of which is a partially corrupted fundamental memory state, which acts as the target final state for the network. m_0 is the proportion of each sample pattern which must be the same as its target state in order that all sample patterns will converge upon their targets. m_1 is the greatest overlap of each sample state with the fundamental memories of the network other than the one on which it is based. Details of the method used for calculating this value are presented in [11].

3.3 Training time

The time taken to train a network is measured in two ways. Firstly, the number of iterations through the training set is measured and secondly, the mean time taken to perform an iteration is calculated. A mean value is calculated for the real training time as the time taken to perform a single iteration changes for some of the models as they near their stopping criteria. Performing both measurements allows us to take into account the differing computational complexity of each learning rule.

Measurements were conducted on a Wintel-compatible PC with a 600MHz AMD Athlon CPU and 128MB of RAM, running Windows 98SE. All simulations were written and run in Java, using the Sun JDK 1.3 with the Hotspot performance engine.

3.4 Network parameters

All networks were of size N=100. Each model was tested by training networks with sets containing 50 random training patterns (i.e. at a loading of α=0.5). The γ distribution analyses were performed on single networks. The R calculations are averaged over 50 networks, and the training time calculations are averaged over 100 networks in each case

4. Results

4.1 γ distribution analysis

Plotting the distribution of the γ_i values allows us to confirm the class to which each network belongs from the shape of the distribution. As might be expected, the two pseudo-inverse approximators have very similar γ distributions, as shown below.

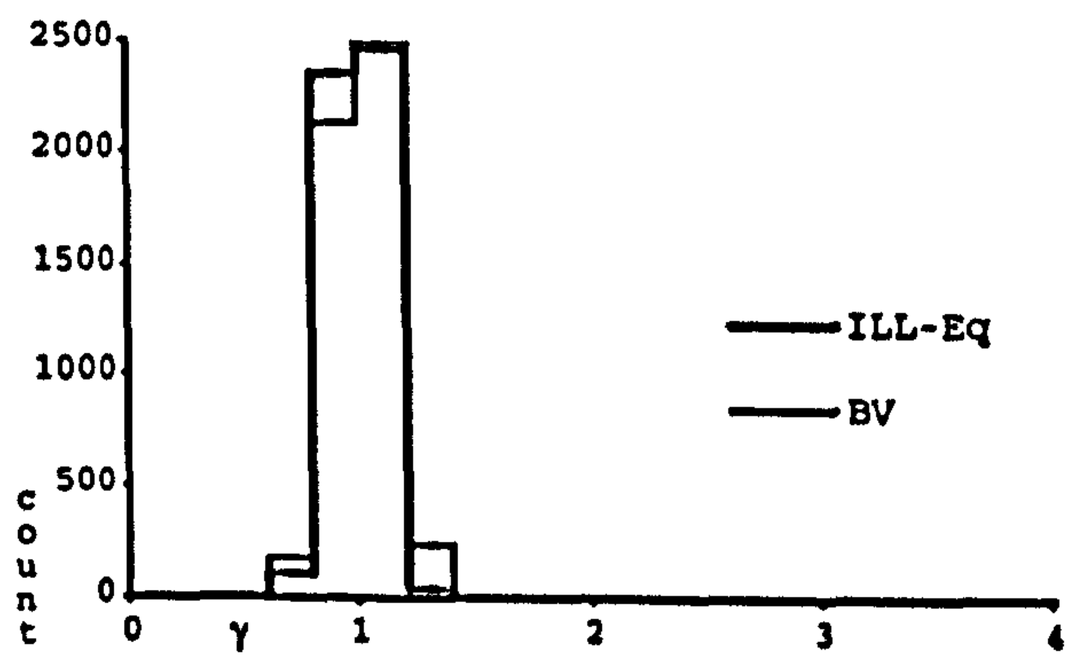


Figure 1. Distribution of γ_i values for ILL-Eq and BV networks.

Whilst it is clear that in neither case are all γ values the same, Figure 1 shows very tight distributions of γ values for the ILL-Eq and BV networks. This is not surprising as both these models are designed to find weight matrices that *approximate* $\Xi\Xi^{\dagger}$. Thus, we place them in Class 2.

Plotting the distribution of the γ_i values for the ILL and KM models gives us:

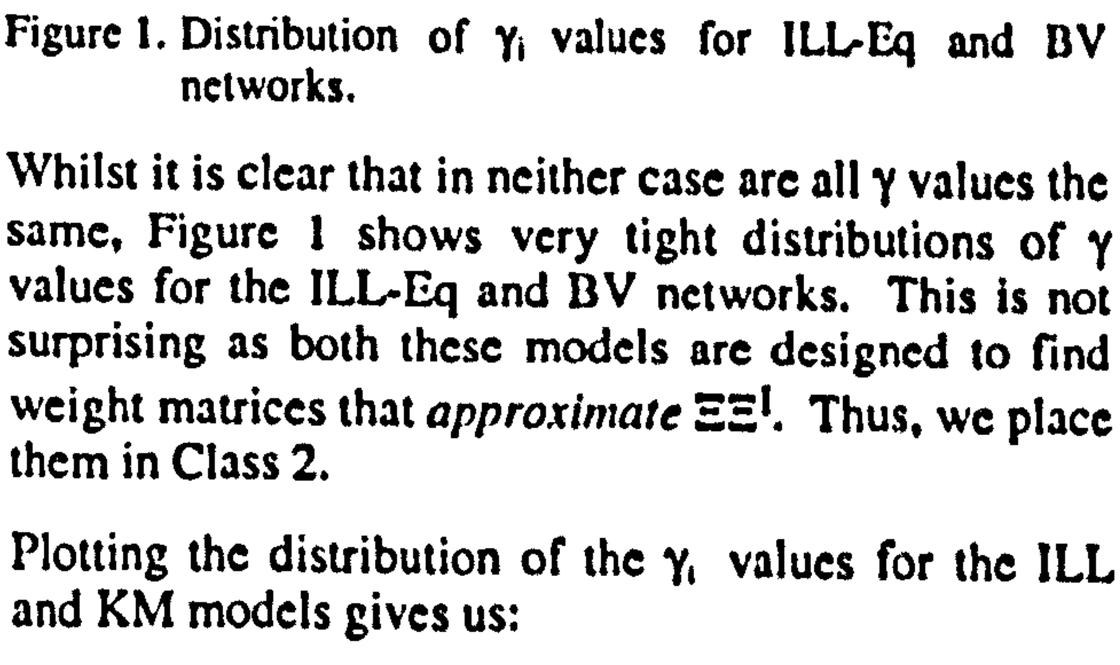


Figure 2. Distribution of γ_i values for ILL and KM networks.

From Figure 2, it is confirmed that the ILL and KM models fall into Class 3.

4.2 Attractor performance

Table 1 shows R values for the different models, including the ILL and KM models at a range of different training thresholds:

Model	R
ILL (T=1)	0.196
ILL (T=10)	0.246
ILL (T=100)	0.262
KM (T=1)	0.253
KM (T=10)	0.254
KM (T=100)	0.270
ILL-Eq	0.215
BV	0.214

Table 1. Values of R for a range of models and parameters

In terms of attractor performance, the model with the highest R and therefore 'best' attractor performance would appear to be KM with T=100.

4.3 Training time

Model	Ave. no. of epochs	Ave. time to train (secs)	Approx. time per epoch (secs)
ILL (T=1)	16.5	1.6	0.1
(T=10)	95.6	9. 8	0.1
(T=100)	895.4	91.4	0.1
KM (T=1)	4.6	18.7	4.1
(T=10)	33.4	144.7	4.3
(T=100)	320.5	1416.5	4.4
ILL-Eq	52.9	18.6	0.4
BV	4.0	2.2	0.6

Table 2. Training times for a range of models/parameters

The first two columns of figures are the mean number of times the training set needed to be presented to complete training, and the mean total training time. The final column is derived from the first two to give an indication of the time taken for each pass through the training set. It should be noted that the KM model does not run through the entire training set in the same way as the other models, so the KM figures represent the time taken to present 50 patterns.

5. Discussion

A number of interesting observations may be made from the above results. Firstly, for the models where a training threshold is used it is clear that R increases with T indicating that this is one means of improving performance, though at the expense of training time. Secondly, the KM rule performs better than the ILL rule, though only marginally so at the higher thresholds. Thirdly, both KM and ILL generally perform better than the pseudo-inverse rules with respect to R, and have a higher maximum capacity ($2N$ vs N).

The KM (T=100) rule seems best if the sole gauge of performance is to be R. However, the time taken to train this network is around 24 minutes (Table 2). Whilst this is not a prohibitively long time, it has to be acknowledged that the increase in R that results when T is changed from 1 to 100 is relatively small, and the trade-off is probably not worthwhile.

The ability of the BV rule to learn new fundamental memories without re-training with the whole training set makes it a better choice for on-line applications. A further investigation of the behaviour of these models is warranted, taking into account such issues as the nature and number of spurious attractor states, in order to determine just how important these relatively small differences in R values are.

6 References

[1] L.F.Abbott, "Learning in Neural Network Memories" *Network: Comp. Neural Sys.* vol.1, pp.105-122, 1990

[2] D.J.Amit, *Modelling brain function: the world of attractor neural networks.* Cambridge, UK: Cambridge University Press, 1989

[3] S.Diederich and M.Opper, "Learning of Correlated Patterns in Spin-Glass Networks by Local Learning Rules" *Physical Review Letters* vol.58, pp.949-952, 1987

[4] N.Davey, R.G.Adams, S.P.Hunt, "High performance associative memory models and symmetric connections", *Proceedings of ISA 2000*, to be published December 2000

[5] W.Krauth and M.Mezard, "Learning algorithms with optimal stability for neural networks" *J. Phys.* vol.A20, pp.L745-L752, 1987

[6] M.G.Blatt and E. G. Vergini (1991). "Neural Networks: A Local Learning Prescription for Arbitrary Correlated Patterns", *Physical Review Letters*, vol.66, pp.1793-1797

[7] L.Personnaz, I. Guyon, and G. Dreyfus, "Collective Computational Properties of Neural Networks: New Learning Mechanisms" *Physical Review A* vol.34, pp.4217-4228, 1986

[8] D. Gorodnichy, "The optimal value of self-connection", *Proceedings of IJCNN'99*, 1999

[9] E.Gardner, "The space of interactions in neural network models" *Journal of Physics* vol.A21, pp.257-270, 1988

[10] I. Kanter and H. Sompolinsky, "Associative Recall of Memory without Errors", *Physical Review A* vol.35, pp.380-392, 1987

[11] N.Davey and S.P.Hunt, "The Capacity and Attractor Basins of Associative Memory Models", *Proceedings of IWANN99*, 1999

Non-Random Weight Dilution in High Performance Associative Memories

S.P Turvey, S.P.Hunt, N.Davey, R.J.Frank

Department of Computer Science,
University of Hertfordshire,
College Lane, Hatfield, AL10 9AB. United Kingdom
s.p.turvey@herts.ac.uk, s.p.hunt@herts.ac.uk, n.davey@herts.ac.uk, r.j.frank@herts.ac.uk

Abstract *The consequences of two techniques for symmetrically diluting the weights of the standard Hopfield architecture associative memory model, trained using a non-Hebbian learning rule, are examined. This paper reports experimental investigations into the effect of dilution on factors such as: pattern stability and attractor performance. It is concluded that these networks maintain a reasonable level of performance at fairly high dilution rates.*

Key-Words Associative Memory, Hopfield Networks, Weight Dilution, Capacity, Basins of Attraction, Perceptron Learning.

1 Introduction

The associative memories examined in this paper are neural networks based around the standard Hopfield architecture [10]. It has been known for some time [1] that it is possible to build networks with superior performance to that of the original model. This improved performance is achieved by replacing Hopfield's one-shot Hebbian learning rule, either with a rule that finds an approximation to the projection weight matrix, or else with a rule that implements perceptron-style learning. (See [5,6,14] for a comparison of performance of different models).

Weight dilution is a technique for reducing the degree of connectivity within what would otherwise be fully-connected networks. Connections are removed after training has taken place (*post-training dilution*). It has even been suggested that an associative memory may be trained by starting with a fully connected network with random fixed weights and systematically removing a fraction of the connections [12].

For one-shot Hebbian learning it is known [13] that capacity drops linearly with the fraction of connections removed.

2 Models Examined

In each experiment we take a network of N units which we train with a set of N–ary, bipolar (+1/-1) training vectors, $\{\xi^p\}$. The N by N weight matrix is denoted by W, and the state (output) of the i'th unit is denoted by S_i

The high-capacity model studied here is a straightforward modification of the standard Hopfield network. The net input, or *local field*, of a unit, is given by:

$$h_i = \sum_{j \neq i} w_{ij} S_j$$

where w_{ij} is the weight on the connection from unit j to unit i. The *next* state of a unit is derived from its local field and its *current* state:

$$S_i' = \begin{cases} 1 & \text{if } h_i > \theta_i \\ -1 & \text{if } h_i < \theta_i \\ S_i & \text{if } h_i = \theta_i \end{cases}$$

where the threshold, θ_i , is normally taken as zero. Unit states may be updated synchronously or asynchronously. Here we use asynchronous, random order updates. These network dynamics and a symmetric weight matrix guarantee simple point attractors in the network's state space.

A training vector, ξ, will be a stable state of the network if the *aligned local fields*, $h_i\xi_i$ are non-negative for all i (assuming all θ_i are zero). Each training vector that is a stable state is known as a *fundamental memory* of the trained network. The *capacity* of a network is the maximum number of fundamental memories it can store. The *loading*, α, on a network is calculated by dividing the number of vectors in the training set by the number of units in the network, N.

2.1 Learning Rules

Two learning rules have been employed in this work. The first approximates the projection matrix generated using the pseudo-inverse rule described by Diederich & Opper [7]. The second is Gardner's perceptron-like symmetric local learning rule [8].

2.1.1 Blatt & Vergini

Blatt & Vergini [3] present a learning rule which takes the form of an iterative method for approximating the projection matrix. The training algorithm is guaranteed to find an appropriate weight matrix within a finite number of presentations of each pattern if such a matrix exists.

The minimum number of presentations of the training set to perform, P, is calculated as being the smallest integer conforming to:

$$P \geq \log_k \left(\frac{N}{(1-T)^2} \right)$$

where k and T are real valued constants such that $1 < k \leq 4$ and $0 \leq T < 1$, and N is the number of units in the network. k is referred to as the *memory coefficient* of the network; the larger it is, the fewer steps are required to train the network. In this work, k=4 and T=0.5 for all networks trained by this rule.

The algorithm is as follows:

BEGINNING WITH A ZERO WEIGHT MATRIX
FOR EACH PATTERN IN TURN
  APPLY THE PATTERN ONTO THE NETWORK
  FOR $m := 1$ TO p
    FOR EACH PROCESSING ELEMENT IN TURN
      UPDATE INCOMING WEIGHTS ACCORDING TO:

$$\Delta w_{ij} = \left(\frac{k^{m-1}}{N} \right) \left(\xi_i^p - h_i \right) \left(\xi_j^p - h_j \right)$$

REMOVE ALL SELF-CONNECTIONS

Note that patterns are added incrementally without corrupting patterns learnt previously.

2.1.2 Symmetric Local Learning

Gardner [9] pointed out that an iterative perceptron-like training rule could be made to produce symmetric weights by simply updating both w_{ij} and w_{ji} when either changes. Gardner also showed that such algorithms would find a symmetric weight matrix, if one existed, for a particular training set.

The symmetric local learning rule is given by:

BEGIN WITH A ZERO WEIGHT MATRIX
REPEAT UNTIL ALL LOCAL FIELDS ARE CORRECT
  SET THE STATE OF NETWORK TO ONE OF THE ξ^p
  FOR EACH UNIT, i, IN TURN
    CALCULATE $h_i^p \xi_i^p$
    IF THIS IS LESS THAN T THEN CHANGE THE WEIGHTS ON CONNECTIONS INTO AND OUT OF UNIT i ACCORDING TO:

$$\Delta w_{ij} = \Delta w_{ji} = \frac{\xi_i^p \xi_j^p}{N}$$

2.2 Weight Dilution

We present two approaches to weight dilution. The first involves the removal of a proportion of the connections chosen at random, the second involves selecting the connections to be removed based upon some heuristic by which it is hoped that the most efficacious connections are retained [2,4].

2.2.1 Random Dilution

A value for the proportion of connections to be removed is chosen. This value is multiplied by the number of connections within the fully-connected network and then halved to give the number of connection pairs to be removed. Then, pairs of units are chosen at random and, if a connection between each pair exists, the bi-directional link is removed. Ensuring that the bi-directional link is fully removed maintains symmetry within the individual connections themselves. This is an important pre-requisite to being able to guarantee that the network will converge upon some stable state when allowed to update freely.

2.2.2 Informed Dilution

Informed dilution operates in much the same way as random dilution in that once a connection is found the bi-directional link is severed. The difference, however, is in the manner in which the connections are chosen. A value for the proportion of connections to be removed is chosen. This value is again multiplied by the number of connections within the fully-connected network and halved to give the number of connection pairs to be removed. Then, the network's connections are scanned to find the smallest weight value (that which is closest to zero). Once the units with the smallest weight value have been identified the connections between them are removed. The process continues until the required number of connections has been eliminated.

3 Analysing Performance

For an associative memory model to be effective, the training patterns should not only be stable states of

the network, but should also act as attractors in the network's state space.

As stated above, the perceptron-type learning rule will store a set of training vectors in the network when the aligned local fields of those vectors have all been driven to be non-negative. Moreover, the larger these aligned local fields become, the better the attractor performance should be. We examine the performance of our networks while varying the loading, α.

We also consider the effect of correlations in the training patterns. An uncorrelated training set is one in which the patterns are completely random. Correlation can be increased by varying the probability that a given bit in a training pattern is +1 (or –1). We refer to the probability of any bit being +1 in each the training vector as the *bias*, b, on the training set. So: $\forall i,p \bullet prob\ (\xi_i^p = +1) = b$, Thus, a bias of 0.5 corresponds to an uncorrelated training set and a bias of 1 corresponds to a completely correlated one, as does a bias of 0.

3.1 Pattern Stability

A pattern is deemed stable if, when applied to the network and the network permitted to run to convergence, the resultant state is equal to the start state. In other words, a stable pattern is a fixed point of the network dynamics.

Pattern stability provides a good indicator of the ability of a particular network to withstand dilution. It is calculated by performing this operation using each of the training patterns as a start state and is given as a proportion of the total number of patterns, i.e. if half the trained patterns are stable then stability=0.5 .

3.2 Attractor Performance

We use, R, the normalized mean radius of the basins of attraction [11], as a measure of attractor performance. It is defined as:

$$R = \left\langle\left\langle \frac{1-m_0}{1-m_1} \right\rangle\right\rangle$$

where m_0 is the minimum overlap an initial state must have with a fundamental memory for the network to converge on that fundamental memory, and m_1 is the largest overlap of the initial state with the rest of the fundamental memories. The angled braces denote an average over sets of training patterns. Details of the algorithm used can be found in [11].

4 Results

In this section we present the results of the performance analyses outlined in section 3. All experiments were carried out on networks of size N=100 trained using patterns of bias 0.5 and 0.9 and at a fixed loading point of α=0.50.

4.1 Pattern Stability

In this section we present the results measuring the stability of the trained patterns while varying the degree of weight dilution within the network.

4.1.1 Blatt & Vergini

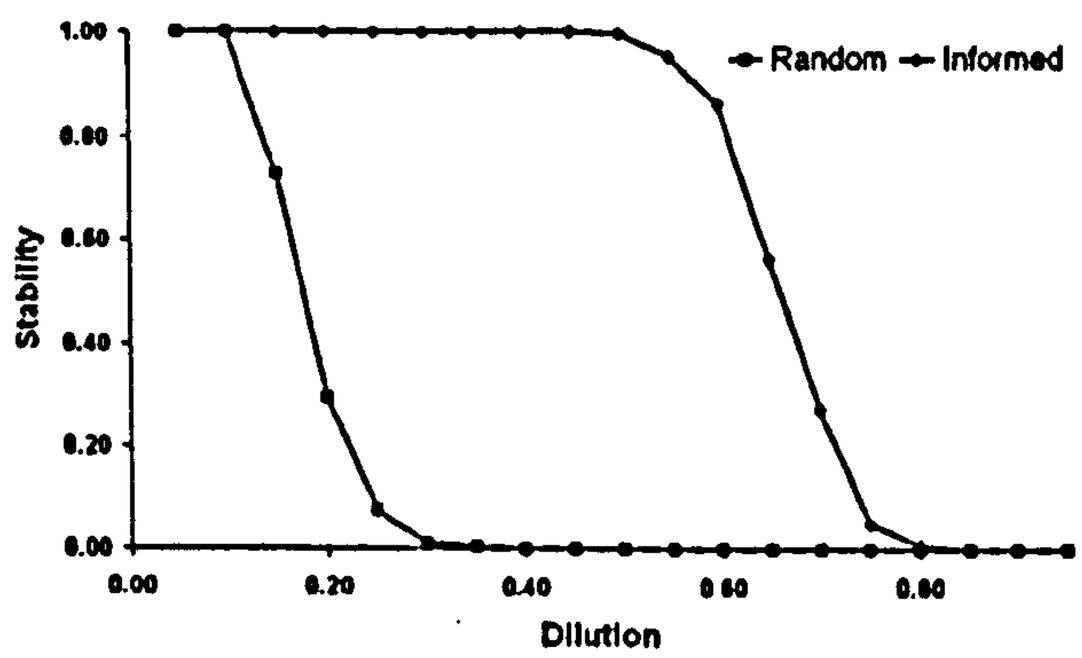


Figure 1: Pattern stability for network trained with Blatt & Vergini under a loading α=0.50 (N=100) using uncorrelated patterns (b=0.5). The upper line represents informed dilution.

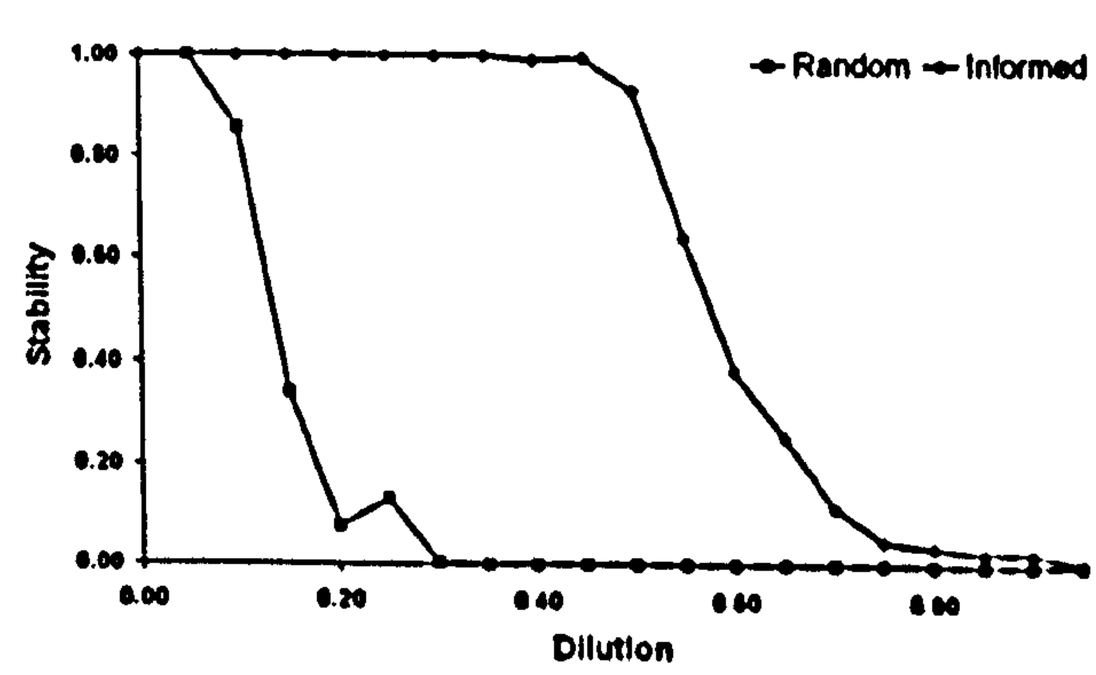


Figure 2: Pattern stability for network trained with Blatt & Vergini under a loading α=0.50 (N=100) using correlated patterns (b=0.9). The upper line represents informed dilution.

4.1.2 Symmetric Local Learning

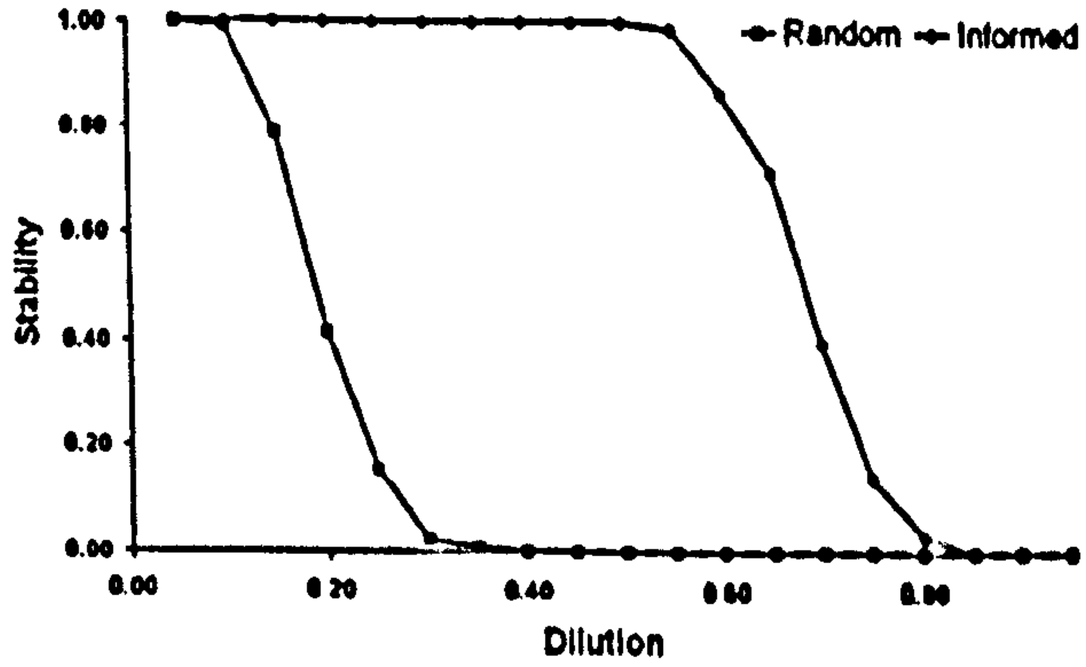


Figure 3: Pattern stability for network trained with Symmetric Local Learning under a loading α=0.50 (N=100) using

uncorrelated patterns (b=0.5). The upper line represents informed dilution.

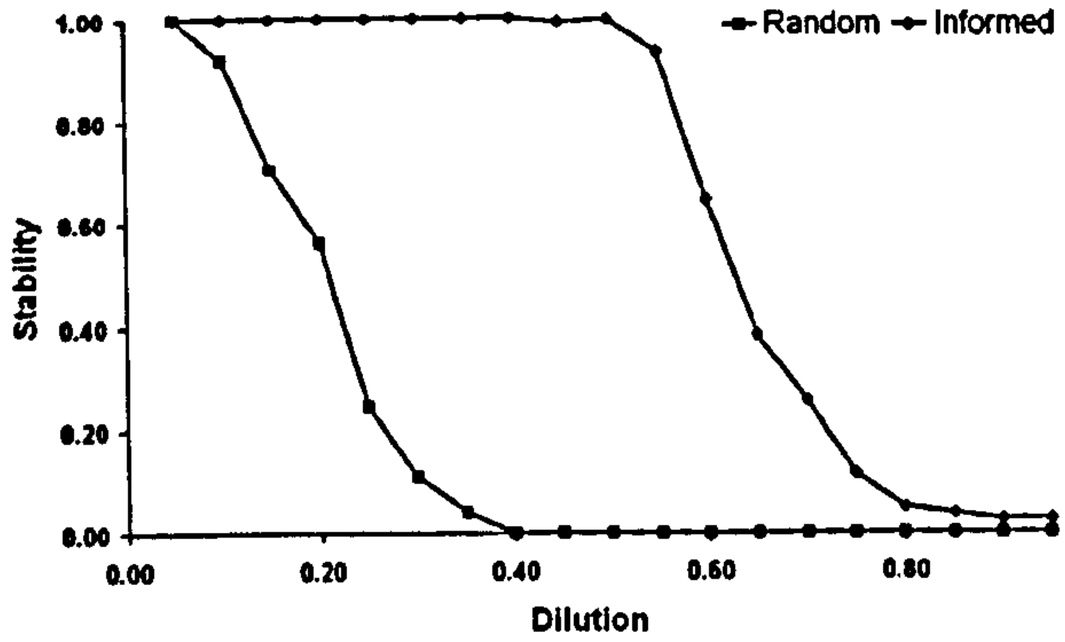


Figure 4: Pattern stability for network trained with Symmetric Local Learning under a loading α=0.50 (N=100) using correlated patterns (b=0.9). The upper line represents informed dilution.

4.1.3 Observations

There are four key observations that can be made from the preceding set of results:

1) Informed dilution gives a clear and significant improvement in pattern stability over simple random dilution. These improvements take the form of an increase in the level of dilution at which the networks retain memory of all the trained patterns.
2) It is possible to remove up to approximately 50% of the networks' connectivity without a serious decline in the stability of the trained patterns.
3) The bias of the trained patterns makes very little difference to the pattern stability. All four plots describe remarkably similar behaviour.
4) The algorithm used, in the case of these experiments, also appears to make very little difference to the effect of dilution on pattern stability.

4.2 Attractor Performance

In this section we present the results measuring the attractor performance of the networks while varying the degree of weight dilution.

4.2.1 Blatt & Vergini

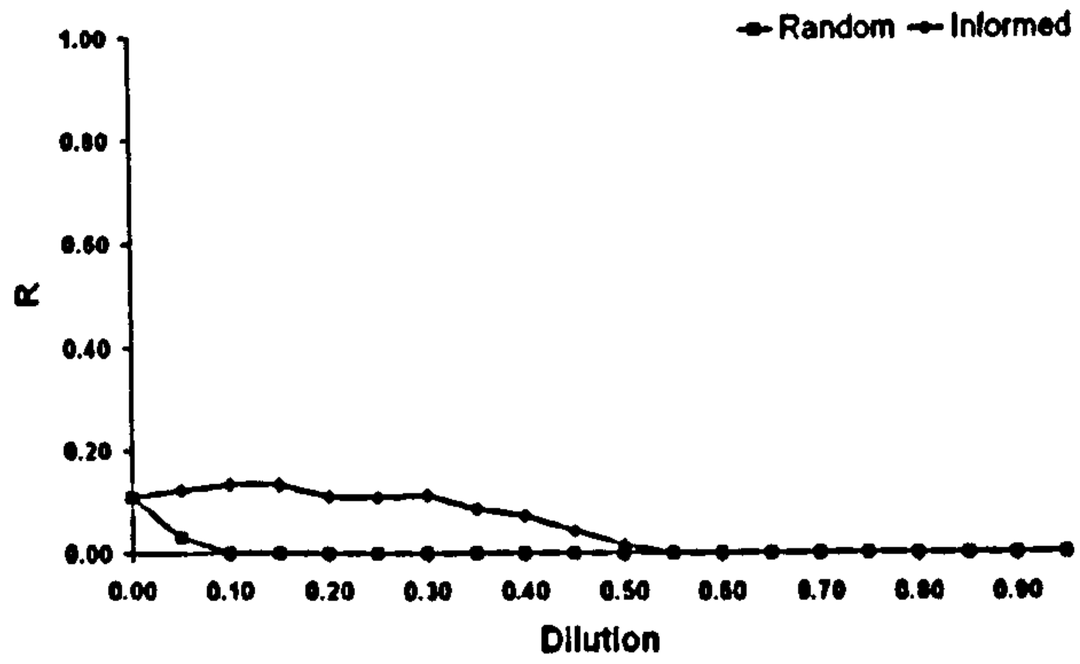


Figure 5: Attractor performance for network trained with Blatt & Vergini under a loading α=0.50 (N=100) using uncorrelated patterns (b=0.5). The upper line represents informed dilution.

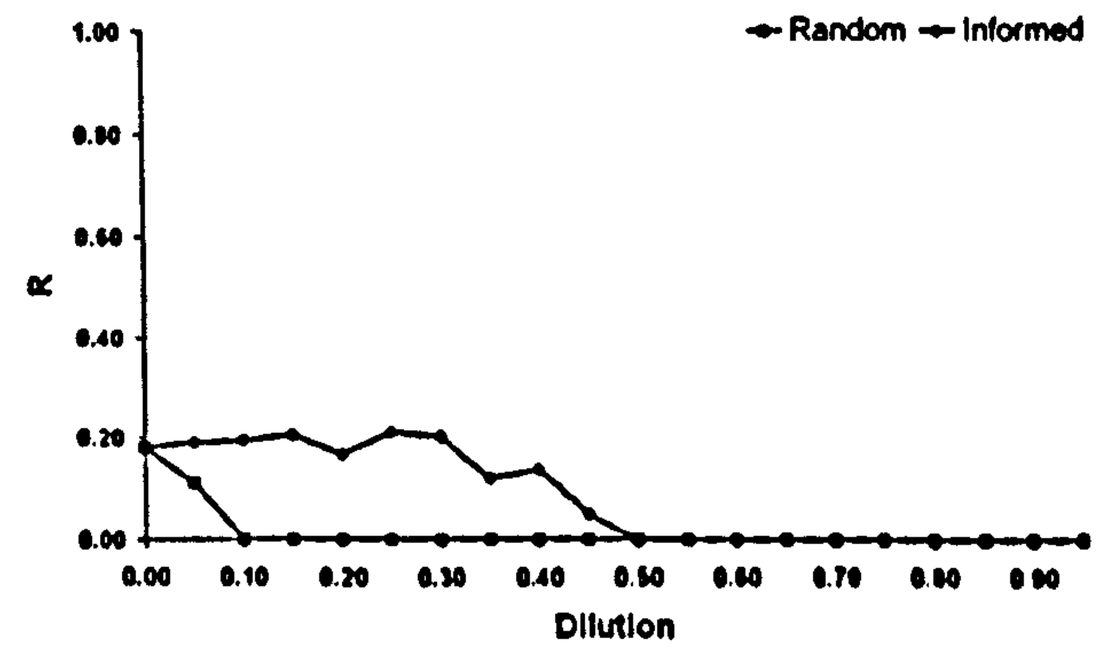


Figure 6: Attractor performance for network trained with Blatt & Vergini under a loading α=0.50 (N=100) using uncorrelated patterns (b=0.9). The upper line represents informed dilution.

4.2.2 Symmetric Local Learning

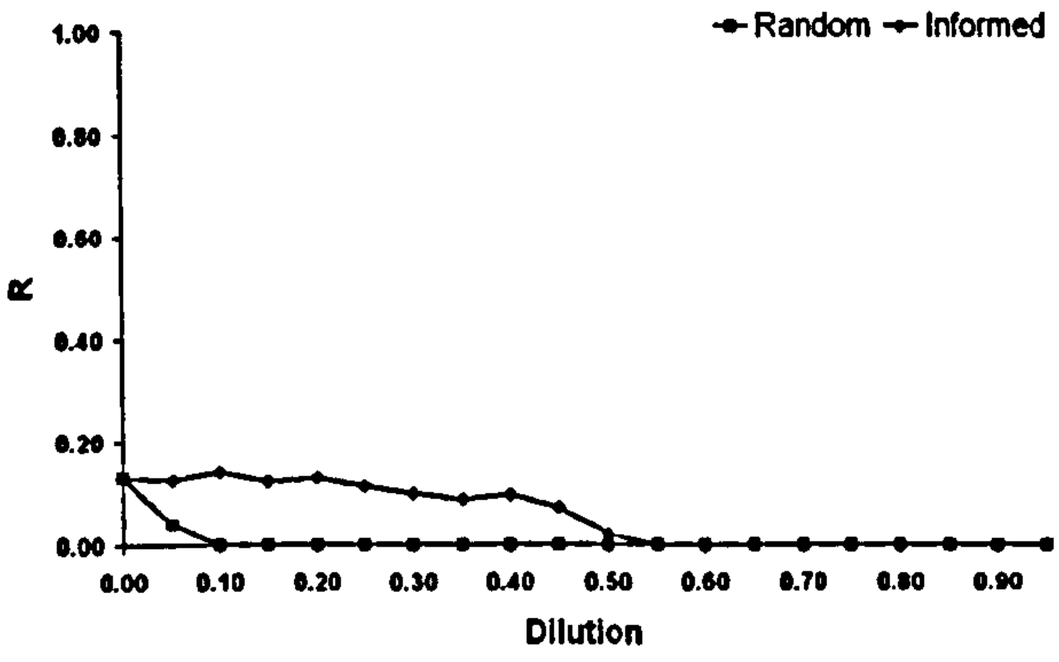


Figure 7: Attractor performance for network trained with Symmetric Local Learning under a loading α=0.50 (N=100) using uncorrelated patterns (b=0.5). The upper line represents informed dilution.

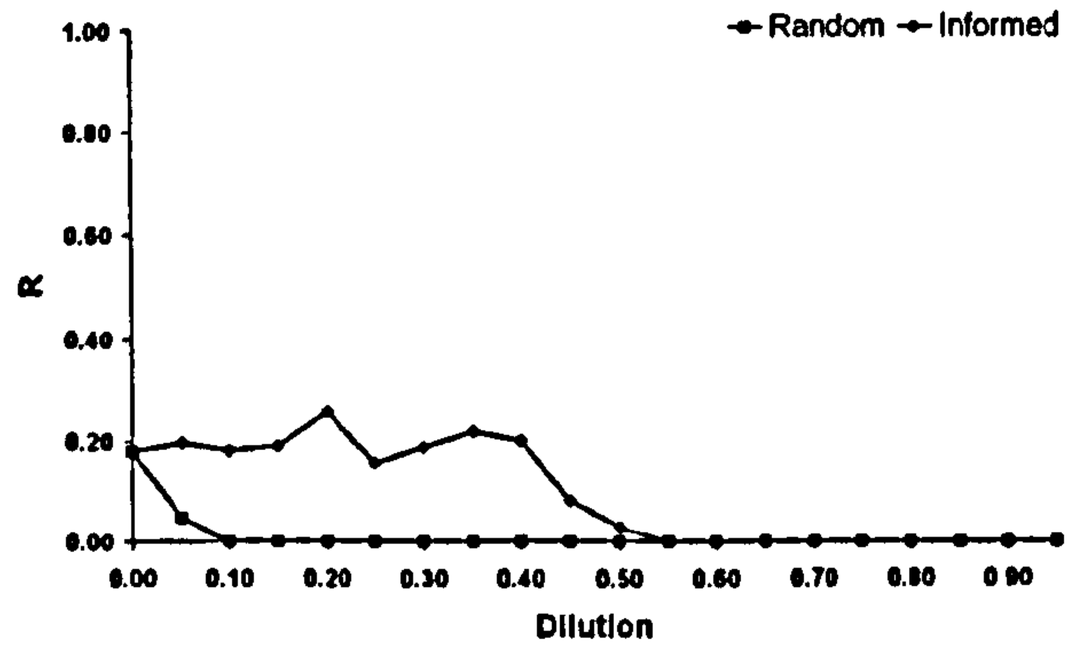


Figure 8: Attractor performance for network trained with Symmetric Local Learning under a loading α=0.50 (N=100) using uncorrelated patterns (b=0.9). The upper line represents informed dilution.

4.2.3 Observations

The pattern of the attractor performance results is similar to that of pattern stability. Specifically:

1) Informed dilution significantly better than simple random dilution.
2) It is possible to remove up to approximately 40% of the networks' connectivity without serious damage to the attractor performance of the network.
3) The bias of the trained patterns makes very little difference to the attractor performance.
4) The algorithm used, in the case of these experiments, also appears to make very little difference to the effect of dilution on attractor performance.

5 Discussion

This paper reports two important results:

1) Informed dilution is markedly better than random dilution.
2) Informed dilution demonstrates that a large number of connections are redundant in networks of this type and at these loadings.

As the loading of these networks is α=0.5 they are below their maximum storage capacity; it may be of interest to repeat these experiments at higher loadings where the networks may be under greater stress with regard their maximum capacity.

It is interesting to note that, for both performance measures, failure, when it occurs, proceeds with great rapidity. There is a sharp decrease in both proportion of stable patterns and attractor performance once the networks begin to lose their stability and ability to act as attractors. In this respect, our results differ from those of Sompolinksy, whose work on randomly diluting the traditional Hopfield network [?] resulted in a linear decline in pattern stability.

The system of informed dilution we have presented is very simple; no re-training of the network is required. It is possible that in biological systems complex strategies may be similarly unnecessary. Chechik *et al* [4] have noted that during brain maturation there is a reduction in connectivity that is expensive to maintain from an energy perspective. It is interesting that our artificial system also demonstrates levels of redundancy in connectivity albeit in a much simpler model.

Our current work has focused on networks that have been created as sparsely-connected *tabula rasa*. Training these networks has presented new challenges and performance characteristics. We hope to be able to present these new findings at a later date.

References:

[1] Abbott, L.F., Learning in neural network memories. Network: Computational Neural Systems, 1990. 1: p. 105-122.

[2] Barbato,D.M.L. and O.Kinouchi, (2000) Optimal pruning in neural networks, *Physical Review E* 62(6), 8387-8394

[3] Blatt, M.G. and E.G. Vergini, Neural networks: a local learning prescription for arbitrary correlated patterns. Physical Review Letters, 1991. 66(13): p. 1793-1797.

[4] Chechik,G., I.Meilijson and E.Ruppin (1998) Synaptic Pruning in Development: A Computational Account, *Neural Computation* 10, 1759-1777

[5] Davey,N. and R. Adams (2001). High Performance Associative Memory Models and Sign Constraints *Proceedings of Neural Networks and Applications (NNA 2001)*, 416-420, 2001

[6] Davey,N. and S.P.Hunt (2000). A Comparative Analysis of High Performance Associative Memory Models. *Proceedings of 2nd International ICSC Symposium on Neural Computation (NC 2000)* 55-61

[7] Diederich,S. and M.Opper (1987). Learning of Correlated Patterns in Spin-Glass Networks by Local Learning Rules. *Physical Review Letters* 58, 949-952

[8] Gardner,E. (1988). The space of interactions in neural network models, *J. Phys.* A **21**, 257-270;

[9] Gardner,E., H.Gutfreund and I.Yekutieli (1989). The Phase Space of Interactions in Neural Networks with definite Symmetry, *J.Phys. A* **22**,

[10] Hopfield, J.J., Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences (USA), 1982. 79: p. 2554-2558.

[11] Kanter, I. and H. Sompolinsky, Associative Recall of Memory Without Errors. Physical Review A, 1987. 35(1): p. 380-392.

[12] Krauth, W., J.-P. Nadal and M. Mezard (1988), The role of stability and symmetry in the dynamics of neural networks, *J.Phys.* A21, 2995-3011.

[13] López,B. and W. Kinzel (1997) Learning by dilution in a neural network, *J.Phys. A* 30 7753-7764

[14] Sompolinsky,H. (1986), Neural Networks with nonlinear synapses and a static noise, *Physics Review* A 34, L519-L523.

[15] Turvey, S.P., S.P.Hunt, N.Davey & R.J.Frank. An experimental assessment of the performance of several associative memory models. in International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA 2001). 2001.