High Capacity Recurrent Associative Memories

N.Davey, S.P.Hunt, R.G.Adams

Department of Computer Science, University of Hertfordshire
College Lane, Hatfield, Herts. AL10 9AB. United Kingdom

Abstract

Various algorithms for constructing weight matrices for Hopfield-type associative memories are reviewed, including ones with much higher capacity than the basic model. These alternative algorithms either iteratively approximate the projection weight matrix or use simple perceptron learning.

An experimental investigation of the performance of networks trained by these algorithms is presented, including measurements of capacity, training time and their ability to correct corrupted versions of the training patterns.

Keywords: Associative memory, Attractor basins, Hopfield neural networks, Learning rules, Perceptron, Performance measures, Pseudo-inverse.

Corresponding Author: Neil Davey, Tel: +44 1707 284310, Fax: +44 1707 284303
e-mail: n.davey@herts.ac.uk

1 Introduction

In this paper we aim to survey the current state of knowledge of a range of high capacity neural network associative memory models, and to present an empirical evaluation of those models. Each model employs a fully connected network of threshold devices, similar to that employed in the original Hopfield model [20].

Networks of the Hopfield type have proved in the past to be amenable to rigorous analysis, and much is known about the behaviour of the standard model under both deterministic and stochastic dynamics [2, 4, 24, 30]. In large part this is due to the strong similarity of such systems to spin glasses, which has enabled some powerful tools of statistical physics to be applied. The standard model uses a very simple weight matrix, formed with a ‘one-shot’ Hebbian rule, which produces a network with relatively poor capacity and performance. A variety of other learning rules have subsequently been proposed which produce better performance, albeit at the cost of increased complexity. The resulting weight matrices are not as easily analysed as those of the original rule and currently some performance questions, for these types of model, are only open to empirical investigation. It is to these questions that this paper is primarily addressed. Other empirical investigations include those of Forrest [15], Krätzschmar & Kohring [26], Schultz [36] and Jagota & Mandziuk [23].

This paper surveys the main types of variation to the basic Hopfield model, and describes a set of empirical tools for evaluating associative memories of this type. The central part of the paper gives the results of an evaluation of the most important features of such networks.

The next Section of the paper discusses Associative Memories in general and motivates the analysis that follows. Section 3 outlines the important features of the models to be evaluated and Section 4 the relevant evaluation methods. Section 5 gives the results and an analysis of them.

2 Associative Memories

An associative memory stores a number of patterns for subsequent retrieval. During recall, the input to an associative memory is used as a *key* to one of the patterns; the output is a *learnt pattern* associated with the key. Suppose each key is an input vector, $\mathbf{x}^P$, and each learnt pattern is a corresponding output vector, $\mathbf{y}^P$: the system is required to learn a set of associations, $\{(\mathbf{x}^1,\mathbf{y}^1),(\mathbf{x}^2,\mathbf{y}^2),(\mathbf{x}^3,\mathbf{y}^3), \ldots\}$, and should respond with the output pattern $\mathbf{y}^P$ every time the network is presented with an input vector, $\mathbf{x}$, which is closer to the pattern $\mathbf{x}^P$ than to any of the other learnt input vectors. Whenever the system gives an output other than $\mathbf{y}^P$ it may be classed as an error:

In functional terms, there are two classes of associative memory:

- *auto-associative*, where each of the learnt associations comprises an input-output pair, $(\mathbf{x}^P, \mathbf{y}^P)$, such that the key *is* the learnt pattern (i.e. $\mathbf{x}^P = \mathbf{y}^P$), and the job of the memory is simply to give as output the learnt pattern that most closely resembles the input it is given;
- *hetero-associative*, where the key is different from the learnt pattern (i.e. $\mathbf{x}^P \neq \mathbf{y}^P$) in one or more of the stored associations. (Typically, $\mathbf{x}^P$ and $\mathbf{y}^P$ will be vectors of different lengths.)

Throughout this paper we concentrate on auto-associative memories.

1.1 Structure and function of memories based upon the Hopfield model

The Hopfield network is probably the best known example of a neural network auto-associator; others include the simple linear associator, the Bi-directional Associative Memory and the Boltzmann machine (see, for example, [19]). It is a fully connected network made up of processing elements (units) that are similar in many respects to bipolar threshold logic units. Here a *fully connected* network is one in which a unit receives inputs from every *other* unit in the network (but usually not from itself).

1.1.1 States of the network

The output of an N unit network at some arbitrary point in time may be represented by a *state vector*, $\mathbf{S}$, where

$$\mathbf{S} = \lfloor S_1, S_2, \ldots, S_i, \ldots, S_N \rfloor,$$

and S_i is the *current state* (output) of unit i, which can be either +1 or –1. The *local field* (net input), h_i of unit i 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*, S', of a unit is a function of its local field and its current state, given by:

$$S_i' = \begin{cases} +1 & \text{if } h_i > \theta_i \\ -1 & \text{if } h_i < \theta_i \\ S_i & \text{if } h_i = \theta_i \end{cases}$$

where θ_i is the threshold of unit i (usually set to zero for all units in the network, but see Section 3.5).

During pattern recall the initial state of the network is set equal to an input vector. The states of individual units are then updated repeatedly until the overall state of the network is stable (i.e. there is no change in $\mathbf{S}$ over time). Units may be updated *synchronously* or *asynchronously* (see Section 2.1.4).

1.1.2 Determining the weight matrix

Each network has an associated N by N weight matrix, which we will denote by $\mathbf{W}$. The contents of this weight matrix are determined by two things: the set of training vectors, $\{\xi^1, \xi^2, \xi^3, \ldots\}$ and the learning rule used to set the weights.

The learning rule employed in the standard Hopfield model is a form of one-shot Hebbian learning, as described in Section 3.2. Whilst the weight matrix generated by this rule has certain desirable properties, the resulting networks have low capacity (Section 2.1.5) and poor attractor performance (Section 4.2). The experimental work presented in this paper is concerned with the investigation of the performance of networks trained under alternative regimes: see Sections 3.2, 3.3 and 3.4 for full details of the various learning rules that we have employed.

1.1.3 The energy function of a Hopfield network

Hopfield [20] describes a function, E, that, given a weight matrix $\mathbf{W}$, ascribes a scalar value to each possible state of the network, $\mathbf{S}$:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} S_i S_j$$

As long as E is bounded and its value decreases monotonically with each update to the state, the network will have a number of locally stable limit points, each of which acts as a *point attractor.* So, given an arbitrary start state, updating proceeds until the network reaches one of the attractor states, at which point it is *stable*, and further updates do not result in any state change. A network of this kind is behaving in a manner analogous to a physical system that is seeking to minimize its energy, so the function E is called the *energy function* of the network.

The existence of point attractors can be guaranteed by ensuring that the network has a symmetric weight matrix and employing an asynchronous update strategy. A network *may* have point attractors in the absence of one or both of these conditions, although there may be other more complex asymptotic behaviours. Such a network is still likely to be useful: indeed, several of the models presented here employ training regimes that give rise to asymmetric weight matrices.

1.1.4 Pattern recall and update dynamics

Each of the system's stable states corresponds to a pattern that is stored in the network. Some of these stable states will be identical to vectors that were in the training set: these are known as the *fundamental memories* of the system. The network will always have other stable states, including the inverses of its fundamental memories. The number of fundamental memories that can be stored, and the number and nature of the additional stable states, depend upon the training algorithm that is employed.

As mentioned above, units may be updated synchronously or asynchronously. Synchronous updating involves calculating local fields and next states for all units before updating them: it is the simplest strategy to implement, but it renders certain state transitions impossible and can result in the appearance of cyclic (as opposed to point) attractors. Under an *asynchronous* update strategy processor activation values are updated one at a time, in either a fixed or a random order. Hopfield suggests updating activation values at random intervals governed by a single parameter - the *stochastic mean update rate*, which expresses the average number of units updated in a given time; however, adopting this strategy makes the detection of stable states unnecessarily complicated, so we employ an alternative, but functionally equivalent, strategy:

```
Repeat until the network reaches a stable state
    Repeat until all N units have been updated
```

```
        Choose one of the units, i, at random and update it
        Remove unit i from the pool of units available for update
    Return all N units to the pool of units available for update
```

By employing asynchronous updating and a symmetric weight matrix ($w_{ij} = w_{ji}$ for all i,j), one can guarantee that the network will only have point attractors, and that the nearest attractor will always be found. However, it *cannot* be guaranteed that all of the network's attractors will be fundamental memories: the *inverse* of each stable state is also stable, some mixtures of the ξ^P may be stable, and there may also be many uncorrelated attractors.

If the network is constrained so that all unit thresholds are zero, it can be seen from Section 2.1.1 that a state, **S**, will be stable if h_i (the local field of unit i) has the same sign as S_i (the current state of unit i) for all i. Equivalently, **S** will be stable if the *aligned local fields*, h_iS_i, are non-negative for all i.

Thus, if we can demonstrate that all aligned local fields are non-negative for every pattern, ξ^P, in the training set, we can be sure that each such pattern is a fundamental memory of the system.

1.1.5 Capacity and loading

The capacity of a network, C, is the maximum number of fundamental memories it can hold. The *loading*, α, of a network is a measure of the size of the training set relative to the number of processing elements in the network and is defined as:

$$\alpha = \frac{\textit{number of patterns}}{N}$$

Although one can place any load upon a network, there is clearly a value for α above which some of the vectors in the training set will not be stored as fundamental memories. We refer to this as the *maximum permissible loading* (or just maximum loading) and denote it by $\alpha_{\max}$:

$$\alpha_{\max} = \frac{C}{N}$$

If this value is exceeded, it is likely that some (possibly even all) of the ξ^P will not be stable states of the network. It has been widely reported [4] that, for the standard Hopfield model, $C \approx 0.14N$, and thus $\alpha_{\max} \approx 0.14$.

1.2 Practical considerations

1.2.1 General

As shown by Abbott and Kepler [3] the learning rules studied in this paper develop weight matrices that fall into three distinct *universality classes.* This means that for large N and high loading, as the networks approach saturation, the behaviour of any one particular model in a particular universality class will act as a general guide to any model in that class.

The importance of universality classes is well explained by Abbott ([2], p108):

> "… network models of associative memory fall into universality classes which may have markedly different behavior away from saturation but which have the same behavior as they approach the saturation limit … When … there are classes of behavior shared by many models it is not essential that the model being studied be a very accurate representation of the real system being modelled. Instead, we must merely require that the model being computed lies in the same universality class as the real system …Also, because of universality, it will suffice to find algorithms which construct one member of each class if we are interested in studying behavior near the saturation limit."

In this paper we present a series of results that illustrate the behaviour of models from each of the three universality classes of recurrent associative memories of the Hopfield type.

Standard Hopfield networks are straightforward to simulate, quick and easy to train, and (relatively) easy to analyse. Unfortunately, however, they have low capacity and poor attractor performance. It has been shown [2, 16] that many of the performance disadvantages associated with Hopfield networks result from the algorithm employed to train them. This paper is concerned with demonstrating those improvements in capacity and attractor performance that may be obtained by modifying the training rule used with networks that adopt the standard Hopfield architecture. Not only is this interesting in itself, but the results are also likely to be informative in the development of high capacity recurrent associative memories with sparse connectivity and non-Hopfield dynamics, such as those intended to model natural neural systems.

1.1.2 Biological plausibility

One of the principal motivations behind the study of recurrent associative memories is the belief that parts of the human brain exhibit similar functionality. A number of workers have used artificial associative memories based upon the Hopfield model in their attempts to explain the mechanisms underlying the operation both of normal and abnormal human memory, including phenomena such as pseudo-rehearsal and unlearning [21, 32]. Hopfield-style networks have also been used in attempts to simulate the behaviour of human memory when subjected to various types of damage, in order to elicit information about the underlying causes of diseases such as Alzheimer's dementia and schizophrenia [8, 22, 33-35].

Whilst it is clear that Hopfield networks have a number of features that make them implausible as models of biological systems, this should not be used as an excuse for ignoring biological plausibility when it comes to evaluating the usefulness of associative memories based upon them. Of particular interest are the connection topology, the rule governing the propagation of activation through the network, and the learning rule. It can be argued that testing the importance, and probing the limitations, of neural plasticity is the fundamental goal of neural network research [2], so we have fixed the network topology and propagation rule for all models under investigation here, and concentrated on investigating the effect of modifying the learning rule.

For a learning rule to be biologically plausible connection weights should be adjusted by a mechanism that relies largely, if not entirely, on local information, because to do otherwise would be to assume that learning in a biological system is directed by an homunculus, or other form of global influence. That is to say that the size of a weight change should depend upon the activation levels (and, possibly, firing order) of the units at either end of the connection, and not upon the overall activation (or error) for the network as a whole. It should also be possible to add new fundamental memories to the system without incurring catastrophic forgetting of the others [32] (as long as the system is operating below its maximum permissible loading). In order to achieve this, a mechanism must be chosen for changing connection weights that employs information that is *local in time as well as in space.* That is to say, the values of a weight change for a connection between any two units, for a particular presentation of a pattern, should depend only upon the activation of those units at that time.

1.1.3 Engineering issues

It might be argued that Hopfield networks are of limited interest from an engineering standpoint: not only do they have low capacity, but there are a number of practical problems associated with them that make them unattractive for many applications. For example, the capacity of this type of network is proportional to the arity of the vectors that it is used to store, and vice versa (see Section 3.1). Thus, if it is necessary to build a bigger network in order to achieve increased capacity, it will also be necessary to modify the vectors that the network is to store. Also, since such networks are fully connected, the amount of memory (and/or the complexity of the wiring) required to implement them increases with the square of the number of units they employ, as does the time taken to train them, and the time they take to reach a stable state during recall. Thus, scaling Hopfield-type networks up in size is not a trivial exercise. Furthermore, it is not always easy to determine whether a network has reached a stable state and, if it has, whether that state is one of the fundamental memories (or, if it is, whether it is the correct fundamental memory).

It is important to note, however, that such models also have certain advantages. For example, because of the use of one-shot Hebbian learning, a standard Hopfield network can be trained very quickly, and it is possible to add new memories to the system incrementally without the need for re-training with earlier memories. Also, the regularity of the network topology makes the design and programming of a Hopfield-type network a trivial task, and renders these systems more mathematically tractable than layered networks with similar functionality.

From an engineering standpoint, therefore, a model should have:
- a regular connection topology (though ideally not full connectivity);
- a fast, local, learning rule that allows for incremental training;
- large basins of attraction for fundamental memories and small basins of attraction for any other stable states;
- few attractors that are not fundamental memories;
- a simple method for determining whether a network has reached a stable state and, if it has, whether that state is one of its fundamental memories.

3 Models Examined

A total of six different models are examined in this paper. Each model adopts the architecture, activation function, output function and update strategy set out in Section 2.1. The models differ from one another with respect to their learning algorithm and their resultant performance. We also examine two types of post-training modification of the networks (see Section 3.6). This section starts with an account of a classification scheme for associative memory models based upon a *normalised stability measure* (see Section 3.1.1), as described in Abbott [2]. The models are then described in the following three sub-sections according to their appropriate classification.

3.1 A classification scheme

Abbott [3] divided all associative memory models into three classes each with a different behaviour and different theoretic maximum storage capacity. Within these classes of models any individual model could have a different behaviour when not near to its saturation point, but all members of a single class converge to the same behaviour as they approach saturation.

3.1.1 Normalised stability measures

As noted in Section 2.1.4, if the aligned local fields for a pattern ξ^p are all non-negative then the pattern is stable. Intuitively, the larger the aligned local fields the larger the domain of attraction. However, consider a network in which every training pattern is stable (so every aligned local field is non-negative for every pattern): any uniform, upward scaling of the weight matrix will increase the aligned local fields, but will obviously not improve the attractor performance. In fact the optimal attractor performance is achieved when the aligned local fields are maximised with respect to the size of the relevant weights [28]. For this reason the relevant characterizations are the *normalised stability measures,* or γ values, defined by:

$$\gamma_i^p = \frac{h_i^p \xi_i^p}{|W_i|} \qquad \text{where} \qquad |W_i| = \sqrt{\sum_{j=1}^{N} \left(w_{ij}\right)^2}$$

The minimum of all the γ_i^p , denoted κ, therefore gives a measure of the overall likely attractor performance:

$$\kappa = \min_{p,i}(\gamma_i^p)$$

Krauth and Mézard [27] show that the value of κ determines the minimum size of the attractor basins in a network. Specifically, if no more than

$$\frac{\sqrt{N}}{2}\kappa$$

bits of a fundamental memory are changed, it can be guaranteed that the network will converge on that memory.

3.1.2 The three Universality Classes

The three universality classes are differentiated according to the distributions of their set of γ values. In Section 4.1 graphs are shown to illustrate the actual distributions obtained by the models considered in this paper.

The first universality class, known as the Hopfield Class since the Hopfield model is the canonical example of this class, has a distribution of γ values given by a Gaussian distribution centred at $\bar{\gamma} = 1/\sqrt{\alpha}$, where $\alpha < 0.14$. The Gaussian distribution will therefore support negative values of γ and give some unstable patterns. Interestingly, Abbott [2] shows that this class has a theoretic maximum loading of 1.14 when the Gaussian distribution is narrower. However no construction is given for the corresponding weight matrix.

Networks in the second class have *pseudo-inverse* weight matrices (see Section 3.3). For a network with such a weight matrix all of the γ values will have the same value, γ_0, given by:

$$\gamma_0 = \sqrt{\frac{1-\alpha}{\alpha}},$$

where α is the loading on the network.

As remarked earlier the behaviour of the models is only exact near saturation so this single value for γ only occurs near the maximum loading of the network. It is clear from the above formula that γ_0 approaches 0 as α approaches 1, and that α (the loading) cannot exceed 1. The third class, known as the Gardner Class (after Gardner's seminal work in this area,[16]), has a distribution of γ values that represents a clipped Gaussian with γ values all greater than zero. The value of κ, which is the minimum value of all the γ values, is therefore also larger than zero and choosing a suitable value allows control over the size of the basins of attraction; the larger that κ is then the better the attractor performance. However, κ is related to the maximum loading, $\alpha_{\max}$, on the network, so that $\alpha_{\max}$ falls as κ rises. Equivalently, for a given loading α there is a corresponding maximum value defined for κ, $\kappa_{\max}$. The relationship between α and $\kappa_{\max}$ is given by Gardner in [16]:

$$\alpha = \frac{1}{\displaystyle\int_{-\kappa_{\max}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \left(\kappa_{\max} + x\right)^2 dx}$$

This relationship is illustrated in Figure 1, below.

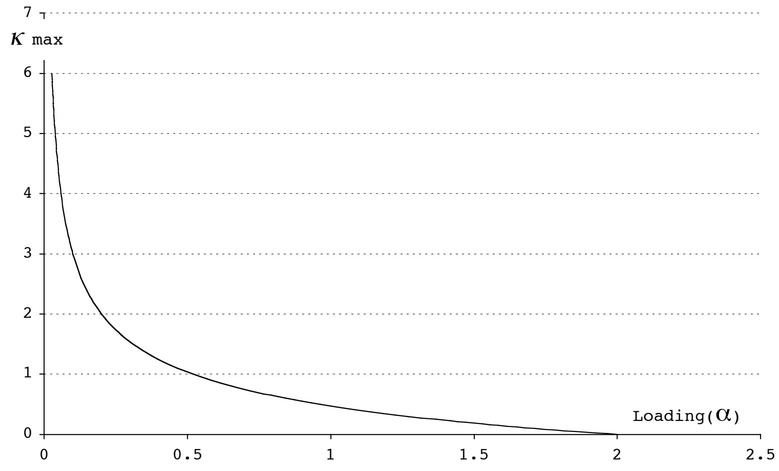

Fig. 1. Graph showing the theoretical relationship between loading (α) on a network and the maximum possible lower bound of the normalised stability measures ($\kappa_{\max}$) for a Class 3 network trained with unbiased random patterns. The higher the value of $\kappa_{\max}$ the better the likely attractor performance of the network.

It is clear from the above graph that the maximum loading for this class of models approaches 2. In fact, as Gardner [16] showed, a network of *N* units should be able to store up to 2*N* uncorrelated patterns, with this optimal capacity increasing for correlated patterns.

It is important to remember that within each class there may be many models with different behaviours away from saturation, but that all members of a single class should exhibit the same behaviour near saturation [1].

3.1.3 Perceptron-style learning

In the late 1980s it was demonstrated that perceptron like learning could be applied to associative memory networks to produce much higher capacity than the basic model [13, 15, 16]. Some of these rules, based on a delta rule that attempts to drive the aligned local fields to a specific value, are an approximation to the pseudo-inverse matrix method. These rules are described in Section 3.4.

Section 3.5 describes other perceptron-style rules, based on a repeated Hebbian technique that is designed to drive the aligned local fields above a learning threshold, *T*. As noted earlier, a necessary and sufficient condition for the training patterns to be learnt is that *T* is non-negative, and often, for ease of training, a value of 1 (or even 0) is taken. Nevertheless, increasing *T* may improve the attractor performance of the network [2, 29], even though it is actually increases in the normalised stability measures, γ_i^p, which take into account the size of the weight matrix, that lead to better performance. However as shown by [2], rules like these give rise to networks in which

$$\kappa \geq \frac{T}{2T+1}\kappa_{\max},$$

where $\kappa_{\max}$ is the optimal value of κ. For positive *T* the right hand side of this inequality is a monotonically increasing function of *T*, so that increasing *T* will in turn increase the lower bound of κ, which may, in turn, give better attractor performance.

3.2 The Hopfield Class of models

Standard Hopfield Learning (HL)

Each connection weight, w_{ij}, is calculated from the training vectors using a variant of the Hebb rule, as follows:

$$w_{ij} = \frac{1}{N}\sum_p \xi_i^p \xi_j^p, \text{ where } i \neq j$$

$$w_{ij} = 0, \text{ where } i = j$$

Since weights may be calculated in a single pass through the training set, this is often referred to as one-shot Hebbian learning.

Storkey Learning (SL)

Storkey [37, 39] has proposed a different learning rule that is also a one-shot rule since it requires only one pass through the training set, and gives a weight matrix that is a first order approximation to the pseudo-inverse (see below). Given an initially zero weight matrix, the addition of a new training pattern, ξ^p, changes weights according to the prescription:

$$\Delta w_{ij} = \frac{1}{N}(\xi_i^p \xi_j^p - \xi_i^p h_j - \xi_j^p h_i)$$

It is worth noting that both HL and SL rules depend only upon information that is local in both space and time: thus, it is possible both to add new memories and to remove existing ones after the initial training has been completed (provided the network's capacity is not exceeded).

In his thesis [38], Storkey gives a further extension to this rule which achieves slightly higher capacity.

3.3 The Pseudo-inverse Class of models

The algorithms employed in this section are all designed to generate weight matrices that are good approximations of the weight matrix generated by the pseudo-inverse rule of Personnaz et al [30]. According to this rule

$$\mathbf{W} = \Xi\,\Xi^{-1}$$

where Ξ is the matrix whose columns are the ξ^P, and Ξ^{-1} is its pseudo-inverse, the matrix with the property that: $\Xi^{-1}\,\Xi = \mathbf{I}$. This weight matrix is also known as the *projection matrix* because it projects onto the sub-space spanned by the training vectors. Such a matrix will embed the training patterns as:

$$\mathbf{W}\Xi = \left(\Xi\,\Xi^{-1}\right)\Xi = \Xi\left(\Xi^{-1}\Xi\right) = \Xi\,\mathbf{I} = \Xi$$

Ξ^{-1} only exists if all of the ξ^P are linearly independent. For random sets of patterns of size less than N this is almost always the case. However, if linear dependencies were to occur, Coombes and Taylor [9] suggest that the weights could be calculated from the eigenvector of the correlation matrix for those patterns.

The models considered in this section belong to the second of Abbott's three universality classes and have a maximum loading, α_{max}, of 1. However, where a network is subjected to a loading in excess of 0.5, single bit flips of the fundamental memories may not be corrected, so the network no longer functions as an associative memory. Kanter and Sompolinsky [24] showed that this loss of performance was due to the presence of self connections (i.e. the projection matrix has a non-zero leading diagonal). They went on to suggest that the weights on these connections could be zeroed without affecting the stability of the fundamental memories, and that the resulting network would perform as an associative memory up to a loading of 1. For this reason, most training algorithms are designed to result in a weight matrix with a zero leading diagonal.

However, the matter is not as simple as it might appear: Gorodnichy [17] has shown that it may be beneficial to employ self connections, as long as the weights on those connections are scaled down in magnitude. The use of pseudo-inverse networks with scaled self connections is discussed further in Section 3.6.2.

Iterative Local Learning with Equal Fields (LL-Eq)

Diederich and Opper [13] propose two perceptron style iterative learning rules. The main model is described in Section 3.5, the one described here is a variant. In this model iteration continues until the aligned local fields for each unit and trained pattern are sufficiently close (within a suitable error margin) to +1. The learning rule is:

```
Begin with a zero weight matrix
Repeat until $\sum_{i,p} \left|1 - h_i^p \xi_i^p\right| < \varepsilon$ , where ε is the maximum permitted error
    Set the state of network to one of the $\xi^P$
    For each unit, i, in turn
    Update the weights to unit i according to:
    $\Delta w_{ij} = \frac{\left(1 - h_i^p \xi_i^p\right) \xi_i^p \xi_j^p}{N}$
```

We take 0.1 as the maximum permitted error.

Diederich and Opper [13] demonstrate that this learning rule will converge for up to N linearly independent training patterns. This model gives an approximation to the pseudo-inverse matrix method where the leading diagonal is automatically already zero. This is, in fact, the delta rule with a learning rate of $1/N$ [19].

Blatt and Vergini's 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 algorithm is as follows:

```
Begin with a zero weight matrix
For each pattern in turn
    Clamp the pattern onto the network and set m = 0
    Repeat until $\sum_{i} \left|1 - h_i^p \xi_i^p\right| < \varepsilon$
        Increment m
        For each processing element in turn
            Update incoming weights according to:
                $\Delta w_{ij} = \left(\frac{k^{m-1}}{N}\right)(\xi_i^p - h_i)(\xi_j^p - h_j)$
```

where k and ε are real valued constants such that $1 < k \leq 4$ and $0 < \varepsilon < 1$, 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, and ε is the maximum permitted error. In this work, $k = 4$ and $\varepsilon = 0.1$ for all networks trained by this rule.

The structure of this algorithm is different from all the other iterative methods described here. Each training pattern is incrementally stored in the weight matrix. Remarkably the addition of each new pattern does not interfere with the storage of the previous patterns.

This algorithm is guaranteed to converge in ν presentations of each input pattern, where:

$$\nu \le \log_k\left(\frac{N}{\varepsilon^2}\right) + 1$$

This model approximates the pseudo-inverse matrix with an intact leading diagonal. Consequently at the completion of the algorithm we need to remove all self-connections. This algorithm differs from the previous one since there is a different update rule, there is a guaranteed maximum number of iterations and since the patterns are added incrementally without interfering with patterns learnt previously.

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 LL-Eq to satisfy its stopping criterion.

1.4 The Gardner Class of models

Iterative Local Learning (LL)

Both Diederich & Opper [13] and Forest [15], proposed using a perceptron-style local learning rule which is an iterative learning rule in which the local fields for each training pattern are driven to the correct side of $+T$ or $-T$ as appropriate. This is equivalent to the condition that:

$$h_i^p \xi_i^p \ge T \quad \text{for all } i,\ p$$

So the 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 $\xi^p$
        For each unit, i, in turn
            Calculate $h_i^p \xi_i^p$. If this is less than T then change the weights to
            unit i according to:
                $\Delta w_{ij} = \frac{\xi_i^p \xi_j^p}{N}$
```

This is the perceptron learning rule with a fixed margin of T and a learning rate of $1/N$. The process will converge on a suitable weight matrix if one exists [13], at which point the trained patterns are guaranteed to be stable. This network has a γ classification in the Gardner Class with a maximum loading, $\alpha_{\max}$, of 2.

Krauth and Mézard's local learning rule (KM)

A modification to the iterative local learning rule proposed by Krauth and Mézard [28] can be shown to produce a κ value that tends towards $\kappa_{\max}$ as T increases. In this version the patterns are not presented to the network in an arbitrary order. Instead the pattern that has the smallest aligned local field is chosen as the one for next presentation.

```
Begin with a zero weight matrix
Repeat until all local fields are correct
    For each unit, i, in turn
        Select the pattern, $\xi^p$, with lowest aligned local field at this unit and
        update the incoming weights according to:
            $\Delta w_{ij} = \frac{\xi_i^p \xi_j^p}{N}$
```

This network is also in the Gardner Class with a maximum loading, $\alpha_{\max}$, of 2.

1.5 Alternative training strategies not investigated in this paper

A number of alternatives have been suggested to the training strategies identified in Sections 3.2-3.4. Two examples that are worthy of note are Athithan's method for finding a Gardner Class weight matrix [5], and Plakhov & Semenov's method for finding a pseudo-inverse weight matrix [31].

Athithan [5] treats the training of the network as an optimization problem to be solved by linear programming methods. Whilst this approach has its merits, the use of such techniques is outside the scope of this investigation, because we are concentrating exclusively on neural network training methods.

Plakhov and Semenov [31] train their networks according to the following algorithm:

```
Initialise the weights using one-shot Hebbian learning
Repeat
    Set the network state to a random configuration
    Update the weights according to:
        $\Delta w_{ij} = -\frac{\varepsilon}{N} h_i h_j$
```

As long as ε is less than a critical value (the reciprocal of the largest eigenvalue of the Hebbian connection matrix), the algorithm converges on a weight matrix which is a scalar multiple of the pseudo-inverse matrix. This method for determining the weight matrix is interesting because it employs only Hebbian terms. However, to make full use of this model, it would be necessary to conduct a detailed investigation into the convergence properties of the training algorithm.

Other Pseudo-Inverse-related associative memory models include those of Brucoli, Carnimeo & Grassi [7] and Yen & Michel [40].

1.6 Post-training modification of networks

1.6.1 Adjustment of unit thresholds

In the normal Hopfield network all unit thresholds are set to zero. In the Adjustable Threshold Network [36] an attempt is made to use the thresholds to maximise the network performance once a weight matrix is in place.

The motivation for this method can be seen from a consideration of a unit, *i*, in a trained network, whose local field for each of four training vectors is: -3, -1, 5 and 7. If *i* has a zero threshold, its state will be –1 for the first two patterns and +1 for the latter two. Schultz [36] proposes maximising the "slack" over the training set, by moving the threshold for each unit so that it gives maximum separation between the positive and negative local fields; in this case that would mean putting the threshold half way between –1 and +5, giving $\theta_i = +2$. In general then, the threshold for a unit, *i*, is set half way between the positive and negative fields with the smallest magnitudes, as given by:

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

If threshold adjustments are performed after training as described above, training patterns that were stable before adjustment should remain stable after adjustment. Thus, variable thresholds should be usable with any learning rule.

In particular, the adjustable threshold method can be used with the Class 1 models. For both Class 2 and Class 3 models, the calculated value of θ_i according to the above formulae is approximately 0, so the movement of the unit threshold is likely to have very little effect (see Section 5.3.2).

1.6.2 Adjustment of weights on self connections: the Gorodnichy rule (G)

Gorodnichy [17, 18] 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). The mechanism employed in our work is to take Blatt and Vergini's approximation to the pseudo-inverse matrix method and, rather than zeroing the main diagonal, to reduce all the values on the diagonal by some fixed factor.

The LL-Eq, BV and G strategies all employ iterative learning algorithms that use local information to generate a weight matrix, $\mathbf{W} = \Xi\,\Xi^{-1}$. The LL-Eq strategy produces a matrix with its diagonal set to zero, the BV strategy sets the main diagonal to zero at the end of its computation and the G strategy reduces the diagonal by a fixed factor.

4 Analysis Tools

4.1 Introduction

Different analysis tools are relevant to different models. However as already described a γ distribution can be computed for each model, and these are illustrated in Section 5.1. For models in the first universality class it is also relevant to calculate the maximum capacity, the other classes having fixed maximum capacities, defined by their class. The learning time, usually measured in epochs, is relevant to the iterative methods. The κ value, and how close it can get to the ideal maximum value $\kappa_{\max}$, defined for the Gardner of models is analysed for the two models in this class. Also the Iterative Local Learning method and the Krauth and Mézard method have a T value associated with them, which the aligned local fields need to be driven above. This value can be varied and results compared. Apart from this all models can be compared using the size of the attractor basins after training and an appropriate measurement for this is described next.

4.2 Attractor basin size

An effective associative memory model is expected, not only to have the training patterns as fixed points of the network dynamics, but also that these fixed points should act as attractors in the state space. The ideal behaviour of such an associative memory would be such that a given initial state should relax to the nearest trained pattern. It is therefore important to know the mean size of the basins of attraction of the trained patterns.

Since the attractor basins cannot be expected to be Hamming hyperspherical [37], it is usual to take the minimum Hamming radius:

$$R\left(\xi^{p}\right) = \inf\left\{\left|\mathbf{q} - \xi^{p}\right| : \mathbf{q} \in Basin\left(\xi^{p}\right)\right\},$$

where $Basin(\xi^P)$ is the set of states that are attracted to ξ^P.

The mean radius of attraction over the patterns, R, can act as a measure of the quality of a particular associative memory. It is also common for R to be normalised with respect to the size of the network, so that it lies between zero and one.

For very small networks it is possible to explore the state space exhaustively (see, for example, [30]), in order to calculate R exactly, but for more realistic network sizes the nature of the attractors is very hard to compute [14, 25] and only empirical methods, as described here, are available.

A sample of states at a fixed distance, r, from a trained pattern, ξ^P, is made, and if all of them relax to ξ^P, it is concluded that $R(\xi^P)$ is at least as big as r. Clearly, the larger the sample size the higher the quality of the estimate, in all of our experiments the sample size is 50. An analysis of the affect of sample size on the estimate of R can be found in [12].

In our implementation we have slightly adapted the method of Kanter and Sompolinsky [24] in the calculation of R. For each of the sample states chosen a fixed fraction, m_0, of the state is identical to the corresponding part of one of the stored patterns, ξ^P , and the rest of the state is random. Initially a low value is taken for m_0 and consequently it needs to be incrementally increased until all of the sample states relax to ξ^P. Averaging the final values of m_0 over different sets of stored patterns yields:

$$R = 1 - \langle m_0 \rangle$$

As is pointed out in [24], for finite size associative memories, another factor needs to be considered. Each of the initial states used in this calculation may overlap one of the other stored patterns more closely than ξ^P, and to compensate for this the definition of R is modified to:

$$R = \left\langle \left\langle \frac{1 - m_0}{1 - m_1} \right\rangle \right\rangle$$

where m_1 is the overlap with the closest of the other stored patterns. The double average is taken over different starting points and different sets of patterns.

So in our implementation, a fixed number of random starting points are chosen, each of which has a low overlap with the members of the training set (low average m_0). If, as is likely, the start state does not relax to the closest training pattern in one or more of the random cases, the value of m_0 is increased (by $1/N$), and the search is repeated. This continues until all random start states relax to the closest stored pattern. This procedure is performed for 50 different sets of stored patterns for each network type.

The perfect attractor network has $R = 1$, which means that it is possible to move away from any stored pattern, and stay within its basin of attraction up to the point at which another stored pattern becomes nearer (see Figure 2). Note that the calculation of average attractor basin size for the trained patterns can only be undertaken when these patterns are themselves stable.

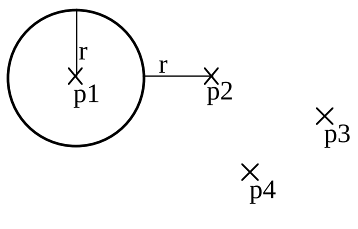

Fig. 2. Calculating R. In this figure p1, p2, p3 and p4 are patterns. The closest pattern in the training set to p1 is p2, at a distance of 2r. Optimal performance occurs when all vectors within the hypersphere centred on p1 and radius r, are attracted to p1. If all patterns stored in a network exhibit this performance, its normalised average basin of attraction, R, is 1

We can also consider the effect of correlations in the training patterns. An uncorrelated training set is one in which the value of each bit in each pattern is chosen completely at random (i.e. *prob* ($\xi_i^p = +1$) = *prob* ($\xi_i^p = -1$) = 0.5, for all i,p). The correlation can be increased by varying the probability that each bit is +1; the probability that a bit chosen at random from a training is +1 is the *bias* on the training set. Thus, a bias of 0.5 corresponds to an uncorrelated training set and a bias of 1 to a completely correlated one. Since we have employed bipolar (+/-1) networks throughout this work, and a training set with bias, b, has the same level of correlation as a training set with bias $1-b$, we present results only for networks trained with sets of patterns with bias 0.5 or greater.

5 Results

In this Section the empirical evaluation of the various associative memory models described in Section 3 is presented. The first Section examines the γ distributions of the models.

5.1 Gamma distributions

The graphs in Figure 3 illustrate the different γ distributions obtained for networks from the three different universality classes. Several 100 unit networks were trained with the same set of 30 unbiased random patterns. The γ values were calculated for each network, then the frequency distributions were plotted. Each training pattern has associated with it 100 γ values – one for each unit in the network – so each chart shows the frequency distribution for 3000 γ values.

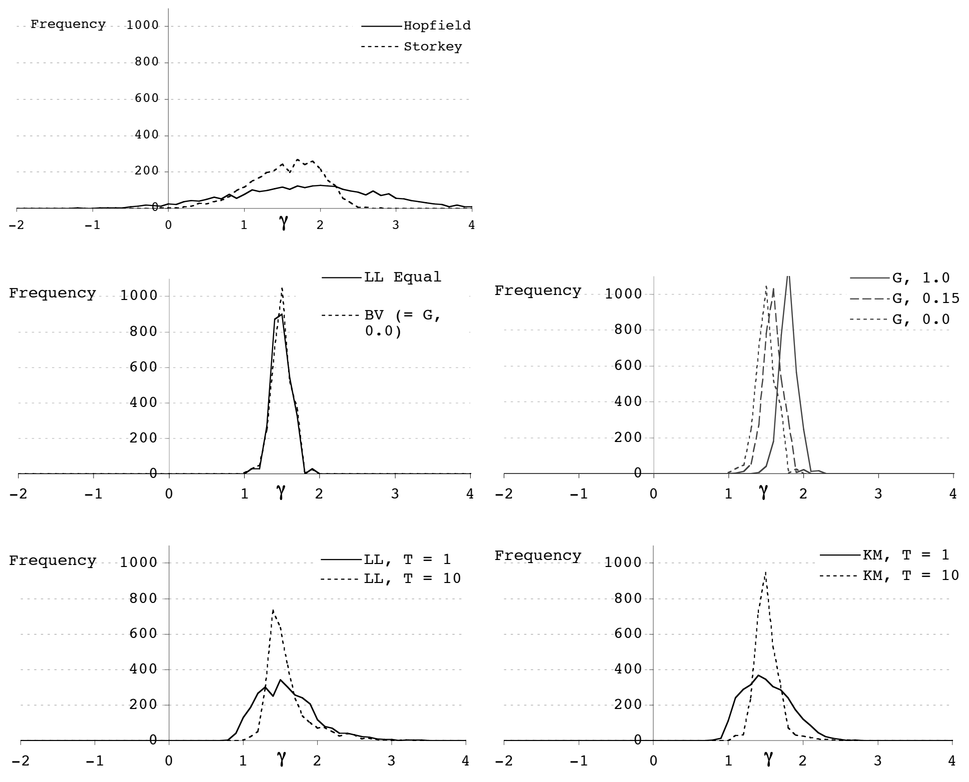

Fig. 3. The γ distributions of the three different classes of associative memories. Results are for 100 unit networks, all trained with the same set of 30 unbiased patterns. For the Gorodnichy networks, G, the dilution rate is also given so that G, 0.0 refers to a network with no self connections. The top row shows results for networks trained with Class 1 rules, whilst the second row shows results for Class 2 rules and the bottom row results for Class 3.

The Hopfield learning rule shows the expected Gaussian distribution. This network has a relatively large number of γ values that are lower than zero (more than 100). This explains why at this loading, $\alpha = 0.3$, the Hopfield learning rule performs poorly. The Storkey rule tightens the distribution, and improves performance. Nevertheless, there are still some negative γ values, so that some of the training patterns are not stable.

The pseudo-inverse (P.I.) rules produce the expected tight distribution. Interestingly the full pseudo-inverse matrix, G, 1.0, which has an unmodified (and hence non-zero) leading diagonal, has the highest minimum γ, whilst diluted matrices have γ distributions further to the left. The local learning rule that approximated the pseudoinverse (LL-Eq) is seen to produce almost the same distribution as the fully diluted P.I. matrix, as expected.

The two rules, LL and KM, both produce the expected clipped Gaussian distributions. The effect of increasing T can be clearly seen: the minimum of the γ values is increased. There is little observable difference in the distribution of γ values between the LL and its optimal version KM.

The next set of results (Figure 4) is for a group of networks trained with a single set of 30 biased patterns (bias = 0.9):

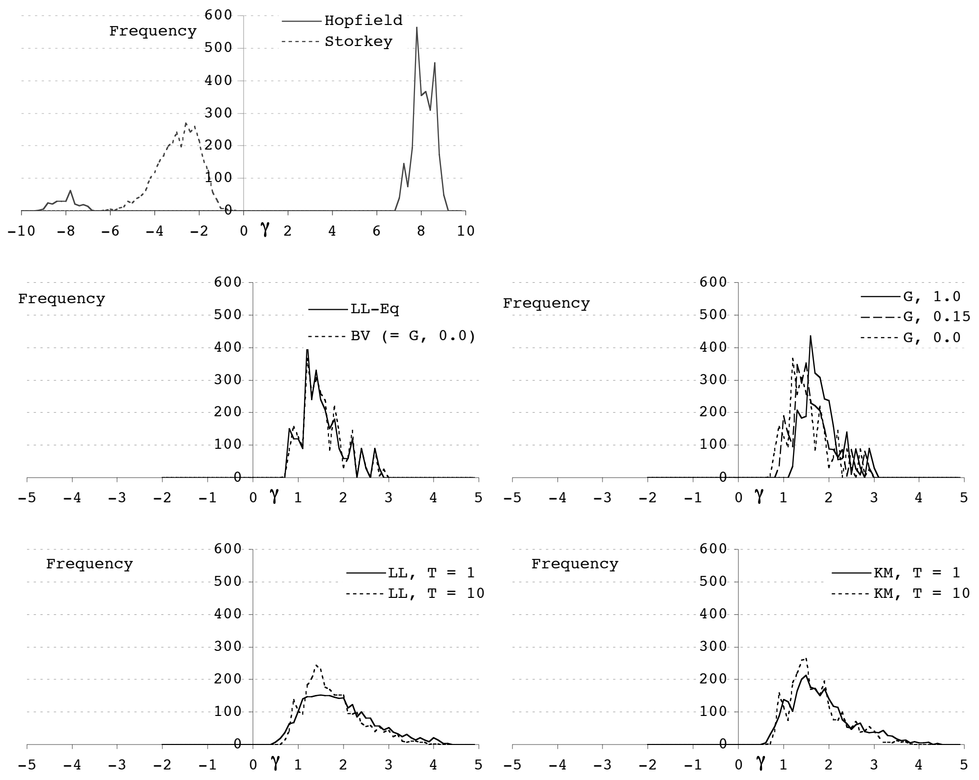

Fig. 4. The γ distributions of the three different classes of associative memories. Results are for 30 biased (bias = 0.9) patterns stored in 100 unit networks. The top row shows results for networks trained with Class 1 rules, whilst the second row shows results for Class 2 rules and the bottom row results for Class 3. Note that these graphs are not drawn to the same scale as those given in Fig.3.

It is apparent that the correlation of the patterns has caused the Gaussian distribution expected for the Hopfeld learning rule to break down: the distribution is bimodal, with many of the γ values being large and negative. The Storkey rule still maintains a Gaussian, but it is much wider than for the unbiased data, and almost all the γ values are negative, so that it is very likely that none of the training patterns are fundamental memories.

The Class 3 rules all produce the expected clipped Gaussian. There does not appear to be much difference between the different versions of these rules.

Finally the pseudo-inverse rules show a less tight distribution for these correlated patterns.

1.2 Capacity

For the Class 1 rules the capacity can be examined. We take a strict interpretation of capacity: the capacity, *C*, of a network is the number of fundamental memories that the network can hold. In practice this can only be approximated. Figure 5, below, shows the proportion of patterns in a training set of unbiased patterns (bias = 0.5) that are stable for the Hopfield network, Storkey network and for a Hopfield network with thresholds adjusted. All results are for 100 unit networks, with averages taken over 50 runs. Adjusting the thresholds cannot ever improve capacity, since any local field that is incorrect will still be incorrect after the threshold is moved. However, moving the threshold can disturb the network dynamics so that point attractors are not reached. This explains the abrupt termination of the line for the Hopfield network with thresholds adjusted, at a loading of 0.14.

The Storkey rule shows a considerable improvement in capacity, with about 22 patterns being storable.

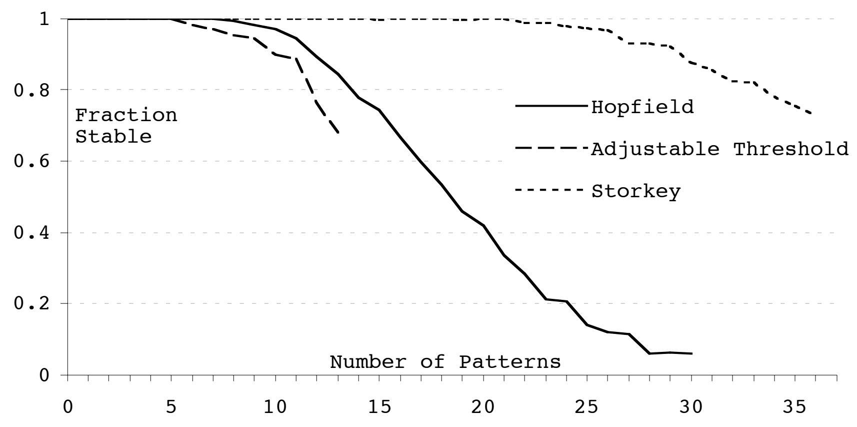

Fig. 5. The capacity of three Class 1 networks of 100 units, using unbiased training patterns. Results are averages over 50 runs.

The capacity of these rules drops with increasing correlation of the training set. This is illustrated in Figure 6. Here the Hopfield network with adjusted thresholds is not shown because it could only handle three patterns before the dynamics broke down.

The other two networks show a markedly worse performance than they did for unbiased patterns, but of the two, the Storkey network has a considerably better capacity (on average about 9 patterns without error),

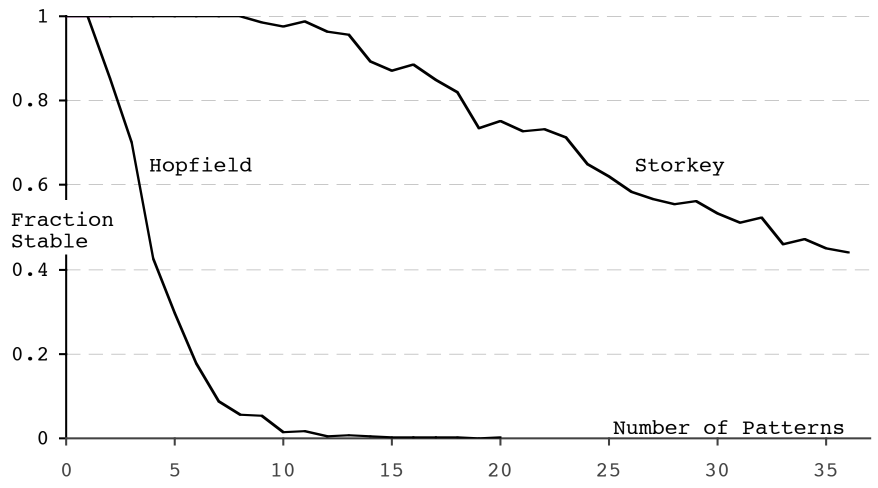

Fig. 6. The capacity of two Class 1 networks of 100 units, using correlated (bias = 0.7) training patterns. Results are averages over 50 runs.

1.3 Attractor Performance

1.3.1 Increasing the loading

Once an associative memory has successfully stored some or all of its training set, initial states should then be attracted to the nearest fundamental memory. As described in Section 4.2, our chosen measure of this property is the mean normalised radius of the basins of attraction of the stored patterns, R. We first consider how R is affected by loading. Figure 7 shows the pattern for local learning: the R values decrease in a roughly linear way as the loading is increased. All other Class 3 rules show a similar pattern, as do the Class 2 rules. Also shown are the R values for the Class 1 Storkey rule, up to the point at which it ceases to embed the training patterns. It can be seen that it is much less effective than local learning. The standard Hopfield rule produces R values close to 1 for training sets of up to roughly 10 patterns (for N=100), at which point it fails to learn all the training patterns.

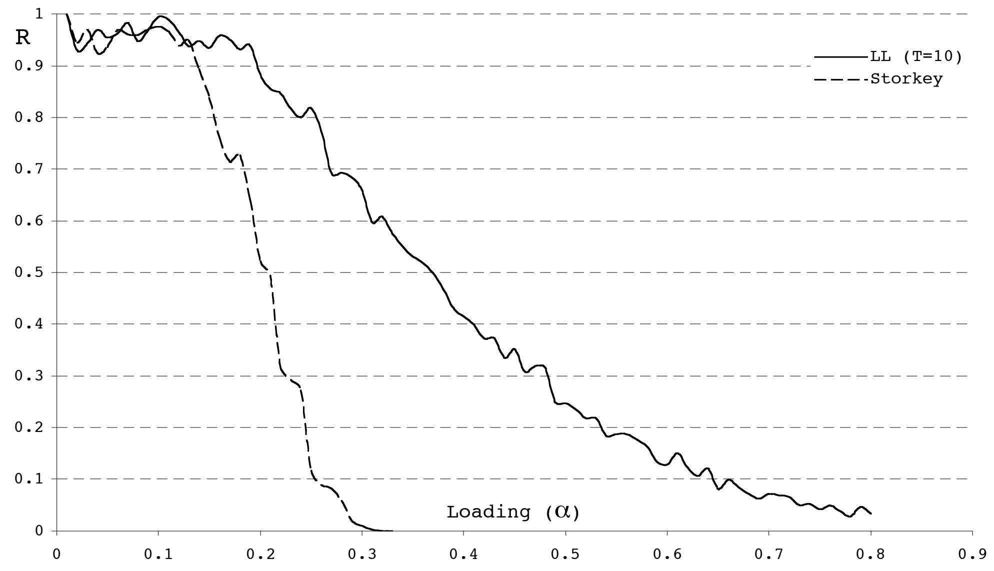

Fig. 7. Attractor performance of local learning (LL) and Storkey rules. For LL the training threshold, *T*, was set to 10. Data are for 100 unit networks trained with sets of unbiased random patterns. Each point plotted on the graph represents the mean of *R* values for 10 different training sets of the same size. Mean *R* values were calculated at loadings ranging from 1 pattern ($\alpha = 0.01$) up to 80 patterns ($\alpha = 0.8$) in increments of 1 pattern.

Figure 8, below, shows how varying the loading (α) on a network trained with the local learning rule affects the value of κ (values are taken from the networks used to measure attractor performance, shown in Figure 7). Also shown is the theoretical relationship between loading and $\kappa_{\max}$. It can be seen that the measured value of κ closely approximates $\kappa_{\max}$ at all loadings.

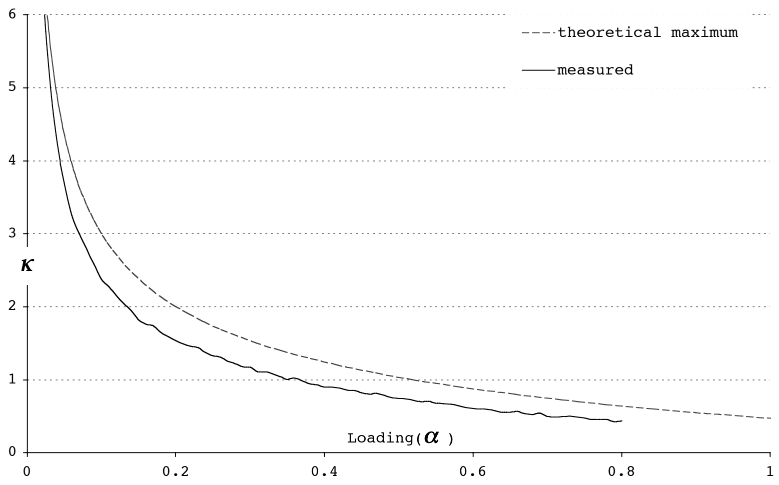

Fig. 8. Graph showing the relationship between κ (the minimum of all the γ values for a network) and loading (α) for 100 unit networks trained with sets of unbiased random patterns using the local learning (LL) rule with a training threshold (*T*) of 10. κ values were calculated at loadings ranging from 1 pattern (α = 0.01) up to 80 patterns (α = 0.8) in increments of 1 pattern. Each loading was tried 10 times. The relationship between the theoretical maximum value of κ and loading (for infinite networks) is also shown for reference.

1.3.2 Varying the learning threshold of Class 3 rules

In order to compare the high capacity rules a more detailed analysis is needed. Table 1 compares the two Class 3 rules, LL and KM, at three different values of the training threshold, T, with 100 unit networks and a loading of 30 patterns. For both learning rules performance improves as T is increased from 1 to 10, but not from 10 to 100. KM produces slightly higher κ values but the *R* values are similar to LL. We also examined the performance of LL 1 with adjusted unit threshold values. This did not make any significant difference to the R value. At this loading, the denominator in the *R* value is about 61, so that the actual mean number of bits corrected is obtained from the reported *R* by multiplying by 0.61, giving between 35 and 39 bits of the 100 in each pattern.

Table 1.
Two Class 3 rules, with training sets of 30, unbiased (bias = 0.5) patterns in 100 unit networks. Each model is trained with three different learning thresholds. The table shows both R and κ, the minimum value of the normalised stability measure. At this loading the theoretical maximum of κ is 1.27. The LL rule with adjusted unit thresholds is also shown. Results are averages over 50 runs.

Model	R	κ
LL 1	0.57	0.84
LL 10	0.64	1.14
LL 100	0.64	1.19
KM 1	0.57	0.89
KM 10	0.64	1.19
KM 20	0.64	1.21
Adj-LL 1	0.58	0.84

1.3.3 Class 2 rules with varying amount of self-connection dilution

Table 2 shows the iterative LL-Eq rule compared with six ‘Gorodnichy’ networks: pseudo-inverse networks trained by the BV rule and with the self-connections diluted by various factors. (*G, 0.0* has the self connections completely removed, *G, 0.10* maintains the self connections at 1/10th their original value, etc)

Table 2
Comparison of Pseudo-Inverse based learning rules with training sets of 30 patterns, unbiased (bias = 0.5), in 100 unit networks.

Model	R
LL-Eq	0.61
G, 0.00	0.61
G, 0.10	0.63
G, 0.15	0.65
G, 0.20	0.64
G, 0.30	0.63
G, 0.50	0.63

The results confirm Gorodnichy’s analysis suggesting 0.15 as the optimal dilution factor. When compared with the Class 3 rules it is interesting to note that the best versions of each produce very similar performance. Specifically LL10 and G, 0.15 have *R* values of 0. 64 and 0. 65 respectively.

1.3.4 Effect of changing correlation of the training patterns

The next set of results (Table 3) shows how attractor performance varies with correlation in the training sets. Just as the capacity of the Class 3 rules increases with correlation in the training set so does the attractor performance. Note, however, that the attractor performance of LL 1 with highly biased patterns (bias = 0.9) is noticeably poorer than LL 10, indicating that the optimum training threshold setting is likely to be influenced by the correlation of the training patterns. Table 4 shows the Class 2 networks with correlated patterns (bias = 0.7). Here the optimal dilution level is not clearly identified as 0.15. Once again the best performance is about the same as the best Class 3 models (R values of 0.79 for both LL10 and G, 0.15 at bias 0.7).

Table 3.
Comparative *R* values for 100 unit networks trained using three different rules. Each model was trained with sets 30 patterns of varying bias. Results are average values obtained from 50 training runs in each case.

Bias	LL 1	LL 10	LL-Eq
0.5	0.57	0.64	0.62
0.6	0.61	0.69	0.64
0.7	0.71	0.79	0.74
0.8	0.87	0.97	0.93
0.9	0.4	0.92	0.8

Table 4:
Pseudo Inverse based learning rules with training sets of 30, correlated (bias = 0.7) patterns in 100 unit networks. The Gorodnichy networks are constructed using the BV pseudo inverse approximator with the self connections progressively kept, so G, 0.0 has the self connections completely removed and G, 0.50 maintains the self connections at half their original value.

Model	R
LL-Eq	0.74
G, 0.00	0.74
G, 0.10	0.78
G, 0.15	0.79
G, 0.20	0.79
G, 0.30	0.80
G, 0.50	0.79

1.3.5 The nature of spurious attractors

In order to attempt to characterise the spurious attractors we performed two experiments, firstly to examine the relative energy of the trained attractors and the spurious attractors, and secondly to examine the overlap of the spurious attractors with the training patterns.

The energy of the attractors in a network was estimated by taking 10,000 randomly chosen initial states and allowing the network to relax to an attractor, whose energy is then calculated. The absolute values of the energies are dependent on the size of the network weights, so that it is not sensible to compare energies across different types of network. Rather, we are interested in the *relative* energies of fundamental memories and spurious attractor states. We take

$$\Phi = \frac{\left\langle E_{fundamental\ memory} \right\rangle}{\left\langle E_{spurious\ attractor} \right\rangle}$$

as our measure of relative energies. All energy values are negative, so a Φ value above 1 shows the fundamental memories to have lower energy than the spurious attractors (on average).

Table 5 shows the results for networks trained with unbiased random patterns. The results are averages of 10 different runs. The first three columns are for 100 unit networks and the final column is for a 500 unit network.

For both of the high capacity rules the energy of the fundamental memories is lower than that of the spurious attractors. As the loading increases the difference in energies is decreased, with both LL and LL-Eq showing very similar values. The much larger 500 unit network shows a very similar pattern of results. For the standard Hopfield network the energy of the spurious attractors is about the same as that of the fundamental memories at a loading of 0.1, but is actually higher at a loading of 0.2.

Table 5
Φ values for networks trained with unbiased random patterns. Each trained network is started in an initial state and allowed to relax to an attractor under deterministic dynamics; this is repeated for 10,000 initial states chosen at random. The results are averages of 10 different training sets, at each loading. The final column is for a 500 unit network trained using local learning. Results are not reported for the Hopfield learning rule at loadings higher than 0.2 as the trained patterns are very rarely stable.

Loading	Hopfield	LL	LL-Eq	LL
	(N=100)	(N=100)	(N=100)	(N=500)
0.1	1.01	1.32	1.33	1.30
0.2	0.85	1.24	1.24	1.22
0.3	-	1.19	1.19	1.16
0.4	-	1.14	1.15	1.11

The second investigation we undertook was to look at the pattern of overlap of a network’s attractors with its training set. The procedure followed was similar to that for the energy experiment above.

A 100 unit network was trained with 10 unbiased random patterns, using the LL rule (T=10). For each individual test, the network was allowed to converge on an attractor. If this was not one of the fundamental memories the overlap between this final state and each of the patterns in the training set was recorded. The results can be seen in Figure 9. Chi-square and Kolmogorov-Smirnov tests did not reveal a significant difference between the observed values and a normal distribution, which is superimposed. Also shown is a normal curve showing the distribution of overlaps between pairs of random patterns. Very similar results were obtained for a network with a pseudo-inverse weight matrix.

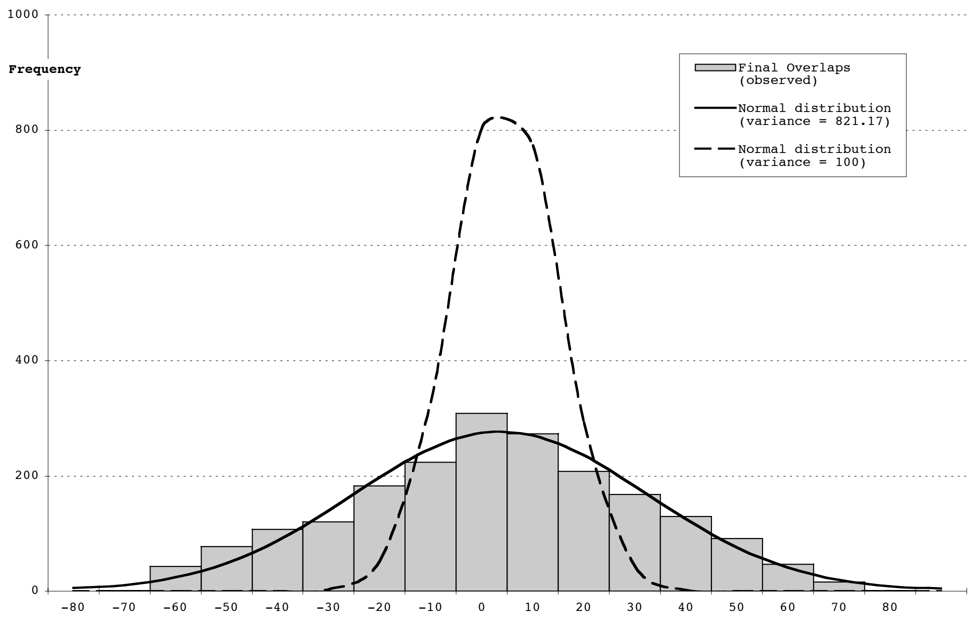

Fig 9: 100 unit network trained using LL and 10 random patterns, 500 random starting points are allowed to relax to an attractor. The distribution is normal and a fitted normal distribution is shown (variance 821.17). Also shown is the normal distribution of overlaps between random pairs of patterns (variance 98.65).

It is apparent that the pattern of overlaps after the network has relaxed is quite different to the random distribution of overlaps. The distribution shows that, often, when a fundamental memory was not recovered, the final state had a stronger correlation with patterns in the training set than could be attributed to chance.

These spurious attractors could be mixture states: odd combinations of training patterns of the form,

$$\xi^{mixture} = sign\ [\pm\xi^{1}\ \pm\xi^{2}\ \pm\xi^{3}...]$$

Consequently, we checked the final states into which the network had settled against all possible mixtures of three training patterns (*triples*), except those that were identical to fundamental memories of the network. This test revealed that stable triples did occur, but with very low frequency: roughly 3 in one hundred recalls. We infer from this that it is likely that only a few of the spurious attractors are pure mixtures.

1.4 Training Times

Several of the rules we have investigated are iterative, and it is therefore relevant to evaluate the time they need to converge on a solution to the training task. The rules that fall into this category are LL (with different training thresholds) and LL-Eq. The BV pseudo-inverse approximator is not iterative in same sense as these rules and is not considered here. Training times are presented as the mean number of epochs needed to produce convergence.

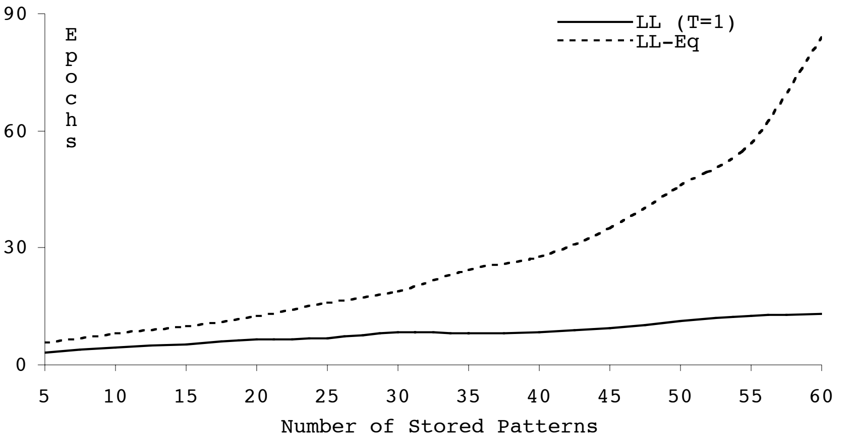

Fig. 10. The number of training epochs required by LL (with a training threshold of 1) and LL-Eq

Figure 10 shows how the number of training epochs is related to loading for LL and LL-Eq. It is apparent that LL has a shallow, linear rate of increase, whereas LL-Eq is following a more pronounced, upward trend.

The next set of results, Table 6, shows how the training threshold and correlation affect training times of LL. It can be seen that increasing T produces a roughly linear increase in training epochs. Making the training vectors more correlated also causes large increases in training times, but the effect of T is similar.

Table 6:
The effect of increasing correlation, and learning threshold on training times for LL. A network of 100 units and loading of 30 patterns. Results are averages over 50 runs.

Model	T	Training Epochs, bias = 0.5	Training Epochs, bias = 0.9
LL	1	7.7	33.6
LL	10	54.8	272.8
LL	100	500.6	2679.6

Table 7 shows how LL and KM perform with a larger training set of 50 unbiased (bias = 0.5) patterns in a 100 unit network. Actual times are also shown. The training epochs increases in an analogous way to the smaller training sets. The KM rule takes longer than LL, showing a similar increase in time with T. KM is a more complex rule than LL and therefore requires more computation per epoch. Note also that BV, the fast method for calculating the pseudo inverse weights, is slower, in real time, than LL10.

Table 7:
The training times of LL and KM rules using a loading of 50 patterns in a 100 unit network. Results are averages over 50 runs. (NA = Not Applicable)

Model	T	Training Epochs	Training Time (s)
LL	1	16.5	1.6
LL	10	95.6	9.8
LL	100	895.4	91.4
KM	1	18.7	18.7
KM	10	144.7	144.7
KM	100	1416.5	1416.5
LL-Eq	NA	52.9	18.6
BV	NA	NA	11.9

1.5 Larger Networks

The final set of experiments examines how well the results already presented scale up to larger networks, namely those made up of 500 units. Table 8 shows that the R values are uniformly higher for this size of network. The number of training epochs required does not increase, rather it decreases. However each epoch now consists of very many more weight updates. The *pattern* of *R* values is repeated; LL 10 has better attractor performance than LL 1 and the pseudo inverse weight matrix with 0.15 of the self connection maintained is better than the matrix with all self connections removed. Moreover the performances of LL 10 and G, 0.15 are once again comparable. Overall the 500 unit networks confirm the *pattern* of results seen in the smaller 100 unit networks, suggesting that the results reported are not sensitive to the size of the networks under consideration.

Table 8:
Number of epochs of training required for 500 unit networks trained with 150 unbiased (bias = 0.5) training patterns, averaged over 30 runs. (NA = Not Applicable)

Model	R	Training Epochs
LL 1	0.78	14.2
LL 10	0.86	57.0
G, 0.0	0.82	NA
G, 0.15	0.85	NA

1.6 Discussion Points

- The analysis of γ distributions confirmed the theoretical predictions and indicated why both the Class 1 rules had difficulty with correlated training patterns.
- The standard Hopfield network is clearly a very poor performer: in fact its poor performance can be predicted very easily by examining its γ distribution graph (see Section 5.1).
- It is clear from the γ distributions that Storkey's attempt to improve on the Hopfield model is only partially successful. For highly biased training sets, where all the γ values are negative for the Storkey network, it is debatable whether there is any improvement at all.
- Adjusting unit thresholds after training, as described in Section 3.5, does not bring any benefit. Indeed it can be seen that such an adjustment causes damage to the simple dynamic behaviour of these networks.
- The Class 2 and Class 3 networks have a very large performance advantage over the one-shot learning rules in Class 1, both in terms of capacity and attractor basin size.
- The best Class 2 model was G, 0.15 (for unbiased patterns), confirming the result of Gorodnichy, although other dilution values may be appropriate for correlated patterns.

- The two Class 3 models examined (LL and KM) gave comparable results, but variation in performance was seen as the learning threshold, *T*, was varied. A value of about T = 10 seems to be most appropriate.
- The performance of Class 2 and Class 3 models was similar. However, LL (T= 10) is probably the best overall model: no other training rule produced better performance and it was faster to converge than other comparable rules. Moreover, it has a simple weight update formulation that relies only on spatially local information, with a capacity of at least 2*N* patterns.

2 Conclusions

In this paper we have set out to present, and empirically evaluate, a range of learning rules for high capacity associative memories. The basic model, originally proposed by Hopfield, is very attractive as an abstract model of associative memory, having point attractors amongst which, at low loading, are the trained patterns. In addition the learning rule is one-shot and local. As shown earlier, Storkey's modification to the learning rule maintains these benefits whilst increasing the capacity.

Two quite different approaches to training these networks have been proposed. The first is to use the pseudo-inverse prescription to create a weight matrix in which the trained patterns are guaranteed to be stable. The second approach uses perceptron training, continuing until all the local fields are correctly aligned and above a specified threshold.

Pseudo-inverse rules only work if the patterns are linearly independent, which for random patterns is very likely, up to the dimension of the weight space, implying that such a learning rule will be able store up to N patterns in an N unit network. Two interactive formulations for calculating the pseudo-inverse matrix, in the context of these types of AMs, have been proposed. The first, LL-Eq, is analogous to the delta rule for training perceptrons, it is local and will converge if a solution exists. The BV learning algorithm is faster and incremental: new patterns can be added to an existing weight matrix without destabilising those patterns already learnt. This property is very important: for all the other incremental learning algorithms, the addition of a new pattern to an already formed weight matrix may produce the phenomenon known as catastrophic forgetting [32], in which significant numbers of previously learnt patterns are destabilized. However, the BV rule relies on a certain amount of non-local information.

The full pseudo-inverse matrix with self-connections has an *effective* capacity of only N/2, as above this loading the trained patterns do not act as attractors (although they are stable). Simply removing all the self connections overcomes this problem, but as Gorodnichy has shown this is not the best policy. In fact maintaining the connections at 0.15 of their original value appears to give optimal attractor performance for uncorrelated training patterns.

The second, perceptron-inspired approach, is exemplified by the LL rule, which simply continues until all the training patterns are learnt. As is well known, perceptron training will converge on a solution if one exists. The capacity of an individual perceptron is known to be at least 2N for a perceptron with N inputs [10], so a Hopfield type network of N units (each of which has N-1 inputs) will have a capacity of approximately 2N using this rule. By changing the order of weight updates, according to the Krauth/Mézard prescription, optimal stability measures can be obtained. Once again this comes at a price: the loss of temporal locality in the algorithm – the entire training set must be considered in order to determine the value of a single set of weight updates for each unit.

These two high capacity variations do produce much more effective AMs. The results Section shows a similar performance for the best versions of these rules. It is notable, however, that the simplest of all these variants, standard perceptron learning (LL), is at least as effective as any other rule; however, it is prone to catastrophic forgetting when additional patterns are added.

A potential difficulty with the Class 3 rules is that they produce non-symmetric weight matrices. The reason for this is that although the weight change on weight w_{ij} is always the same as the weight change on w_{ji}, the two weight changes do not necessarily co-occur in any particular epoch. The presence of asymmetric weights immediately complicates the dynamics of the network – there is no energy function and complex behaviour may arise. Nevertheless if the weights are reasonably symmetric, as they appear to be for the Class 3 rules at the loadings considered here, then the behaviour is close to that of the symmetric networks. Nonetheless it is straightforward to force the Class 3 rules to produce symmetric weight matrices and we have examined the consequences of this elsewhere [11].

From the results presented here it is clear that it is possible to make significant improvements to the performance of recurrent associative memories by modifying the learning algorithm they employ. However, the network models we have investigated still lack certain of the key features that are associated with biological associative memories, such as structured network topologies, sparse connectivity and non-deterministic dynamics. It is our intention to use the results presented here to inform the design of associative memories that are both more biologically plausible and of greater practical use.

3 References

1. Abbot, L.F. and T.B. Kepler, *Universality in the space of interactions for network models*. Journal of Physics A: Mathematical and General, 1989. **22**: p. 2031-2038.
2. Abbott, L.F., *Learning in neural network memories.* Network: Computational Neural Systems, 1990. **1**: p. 105-122.

3. Abbott, L.F. and T.B. Kepler, *Universality in the space of interactions for network models*. Journal of Physics A, 1989. **22**: p. 2031-2038.
4. Amit, D.J., *Modelling brain function: the world of attractor neural networks*. 1989, Cambridge, UK: Cambridge University Press.
5. Athithan, G., *A comparative study of two learning rules for associative memory*. Pramana - Journal of Physics, 1995. **45**(6): p. 569-582.
6. 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.
7. Brucoli, M., L. Carnimeo, and G. Grassi. *Discrete-time cellular neural networks for associative memories: a new design method via iterative learning and forgetting algorithms*. in *38th Midwest Symposium on Circuits and Systems*. 1995.
8. Brunel, N. and J.-P. Nadal, *Modeling memory: what do we learn from attractor neural networks?* Sciences de la Vie, 1998. **321**: p. 249-252.
9. Coombes, S. and J.G. Taylor, *Using Features for the Storage of Patterns in a Fully Connected Net*. Neural Networks, 1996. **9**(5): p. 837-844.
10. Cover, T.M., *Geometrical and Statistical Propeties of Systems of Linear Inequalities with Application in Pattern Recognition*. IEEE Transactions on Electronic Computers, 1965. **14**: p. 326-334.
11. Davey, N., R.G. Adams, and S.P. Hunt. *High Performance Associative Memory Models and Symmetric Connections*. in *International ICSC Congress on Intelligent Systems and Applications (ISA 2000)*. 2000.
12. Davey, N. and S.P. Hunt. *A Comparative Analysis of High Performance Associative Memory Models*. in *2nd International ICSC Symposium on Neural Computation (NC 2000)*. 2000. Berlin.
13. Diederich, S. and M. Opper, *Learning of Correlated Patterns in Spin-Glass Networks by Local Learning Rules*. Physical Review Letters, 1987. **58**(9): p. 949-952.
14. Floréan, P. and P. Orponen, *Attraction radii in binary Hopfield nets are hard to compute*. Neural Computation, 1993. **5**: p. 812-821.
15. Forrest, B.M., *Content-addressability and learning in neural networks*. Journal of Physics A, 1988. **21**: p. 245-255.
16. Gardner, E., *The space of interactions in neural network models*. Journal of Physics A, 1988. **21**: p. 257-270.
17. Gorodnichy, D.O. *The optimal value of self-connection*. in *International Joint Conference on Neural Networks (IJCNN'99)*. 1999. Washington, DC.
18. Gorodnichy, D.O. and A.M. Reznik, *Increasing attraction of pseudo-inverse autoassociative networks*. Neural Processing Letters, 1997. **5**(2): p. 123-127.
19. Hertz, J., A.Krogh and R.G.Palmer, *Introduction to the Theory of Neural Computation*. 1991: Addison-Wesley.

20. 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.
21. Hopfield, J.J., D.I. Feinstein, and R.G. Palmer, *Unlearning has a stabilising effect in collective memories*. Nature, 1983. **304**: p. 158-159.
22. Horn, D. and N. Levy, *Neuronal-based synaptic compensation: a computational study in Alzheimer's disease*. Neural Computation, 1996. **8**(6): p. 1227-1243.
23. Jagota, A. and J. Mandziuk, *Experimental study of perceptron-type local learning rule for Hopfield associative*. Information Sciences, 1998. **111**(1998): p. 65-81.
24. Kanter, I. and H. Sompolinsky, *Associative Recall of Memory Without Errors*. Physical Review A, 1987. **35**(1): p. 380-392.
25. Kepler, T.B. and L.F. Abbott, *Domains of attraction in neural networks*. Journal de Physique France, 1988. **49**: p. 1657-1662.
26. Krätzschmar, J. and G.A. Kohring, *Retrieval dynamics of neural networks constructed from local and non-local learning rules*. Journal Physique de France, 1990. **51**: p. 223-229.
27. Krauth, W. and M. Mezard, *Learning algorithms with optimal stability in neural networks*. Journal of Physics A: Mathematical and General, 1987. **20**: p. L745-L752.
28. Krauth, W. and M. Mézard, *Learning algorithms with optimal stability for neural networks*. Journal of Physics A, 1987. **20**: p. L745-L752.
29. Krauth, W., J.-P. Nadal, and M. Mézard, *The roles of stability and symmetry in the dynamics of neural networks*. Journal of Physics A, 1988. **21**: p. 2995-3011.
30. Personnaz, L., I. Guyon, and G. Dreyfus, *Collective Computational Properties of Neural Networks: New Learning Mechanisms*. Physical Review A, 1986. **34**(5): p. 4217-4228.
31. Plakhov, A.Y. and S.A. Semenov, *Neural networks: iterative unlearning algorithm converging to the projector rule matrix*. Journal Physique de France, 1994. **4**: p. 253-260.
32. Robins, A. and S. McCallum, *Catastrophic forgetting and the pseudorehearsal solution in Hopfield-type networks*. Connection Science, 1998. **10**(2): p. 121-135.
33. Ruppin, E., *Neural modeling of psychiatric disorders*. Network: Computational Neural Systems, 1995. **6**: p. 636-656.
34. Ruppin, E. and J.A. Reggia, *A neural model of memory impairment in diffuse cerebral atrophy*. British Journal of Psychiatry, 1995. **166**: p. 19-28.
35. Ruppin, E. and J.A. Reggia, *Seeking order in disorder: computational studies of neurologic and psychiatric diseases*. Artificial Intelligence in Medicine, 1998. **13**: p. 1-12.

36. Schultz, A., *Five Variations of Hopfield Associative Memory Network*. Journal of Artificial Neural Networks, 1995. **2**(3): p. 285-294.
37. Storkey, A. and R. Valabregue, *The basins of attraction of a new Hopfield learning rule*. Neural Networks, 1999. **12**(6): p. 869 - 876.
38. Storkey, A.J., *Efficient Covariance Matrix Methods for Bayesian Gaussian Processes and Hopfield Neural Networks*, in *Electrical Engineering*. 1999, Imperial College London: London. p. 138.
39. Storkey, A.J. and R. Valabregue, *Hopfield Learning Rule with High Capacity Storage of Correlated Patterns*. Electronics Letters, 1997. **33**(21): p. 1803-1804.
40. Yen, G. and A.N. Michel, *A Learning and Forgetting Algorithm in Associative Memories: Results Involving Pseudo-Inverses*. IEEE Transactions on Circuits and Systems, 1991. **38**(10): p. 1193-1205.