Short-term Associative Memories

Richard Henson

MSc IT:IKBS Project Report

Department of Artificial Intelligence

University of Edinburgh

September 1993

Abstract

Several suggestions for introducing explicit forgetting in artificial neural networks have been studied for Willshaw Net and Hopfield Net models of distributed, associative memory. Such forgetting allows a network to function as a short-term memory, or “palimpsest”. Then continuous learning does not result in eventual catastrophic failure of the memory, but rather the effective storage of a well-defined number of recent memories, accompanied by the progressive forgetting of older ones. The suggestions have been implemented in sizeable networks and their performances compared. They have also been studied mathematically and briefly reviewed from physiological, psychological and implementational perspectives.

Acknowledgements

I would like to thank both my supervisors, Dr David Willshaw and Dr Peter Ross, for sharing their experience through guidance and advice. I am particularly grateful to Dr Willshaw for his time in our weekly project meetings, in which he supplied much enthusiasm and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of Project

The advent of artificial neural networks has allowed computational modelling of psychological theories of distributed, associative memory. One characteristic of these models is a limited capacity for information storage which, when exceeded, typically results in complete failure of the model as a memory device. There is no doubt that animal memories have a finite capacity, but such sudden failure of all memories is never observed. This suggests that an important part of animal memory involves "freeing" capacity by some *forgetting* of information.

Irrespective of psychological modelling, there may well be practical situations where a neural network is required to temporarily store indefinite amounts of new information. This can only be achieved by replacing old information with new, as effectively and efficiently as possible.

The objective of this project is then to examine suggestions for incorporation of forgetting in neural networks. The objective has been met by the implementation of seven main proposals: four identified in recent research papers and three proposed by the author. In the literature, the proposals have been framed in the context of two basic neural network models: the Willshaw Net and the Hopfield Net. The end result is three proposals for the Willshaw Net and four for Hopfield Nets.

Simulation data have shown all methods to be effective, though they differ in theoretical and practical capacities, and also in biological plausibility. Comparison of the proposals has involved their model-independent classification and engendered a more general understanding of the nature of forgetting in distributed memories.

1.2 Background

1.2.1 Associative Memories

From psychological theories of human and animal learning has come the idea of associative memories. These store information by learning correlations between different stimuli. A high correlation between two stimuli means that, when one is presented as a memory *cue*, the other is likely to be retrieved as a consequence: the two have become *associated* with each other in memory.

This correlation has two aspects. One is the frequency with which the two stimuli co-occur during learning (as in classical conditioning). This aspect is not relevant here, since the interest shall be confined to the amount learnt in one presentation of stimuli (so-called *one-shot* learning).

The second aspect involves the nature of the stimuli themselves. By considering them as comprised of a number of component features, correlations can be learnt between some features, but not others. It might be that common features are "reinforced", whereas features not shared by both stimuli are "ignored". Then the stored information is *distributed* in the sense that it is shared amongst numerous correlations between individual features.

Distributed, associative memory can be contrasted to other types of memory, such as that found in a conventional computer. Here, information is not gradually learnt from co-occurrences of data, but is simply "placed" in a number of specially set-aside locations by an inbuilt routine. Each location has a particular *address*, and information is retrieved by returning the contents of a given address. However, the address and its contents need not share any features in common.

Since it is the actual nature of the cue that is important in associative memories, not an arbitrary address, the former are sometimes said to exhibit *content-addressibility*. To see this, consider a case where the cue is not completely correct. In a conventional computer, an incorrect address would simply result in the wrong piece of information being retrieved. In a distributed memory on the other hand, it may still be possible to retrieve the information if the cue is sufficiently similar to the associated stimulus.

1.2.2 Neural Networks

Neural Networks, or Connectionist Models, offer an architecture in which the natural form of memory is distributed. A typical *net* consists of a number of *units*, each of which can be thought as representing a particular feature. The presence, absence, or "degree" of a feature in a particular stimulus then corresponds to the *activation* of that unit. A stimulus maps to a *pattern* of activity over the units.

Associations are learnt by changing the strength of *connections* between units. The connections determine how unit activations interact; specifically their *weights* determine how much activation is "passed" from one unit to another. Learning typically involves presenting stimuli, and then strengthening weights between units with similar activations. Retrieval involves presenting one stimulus and passing activations between units until a pattern of activation representing the associated stimuli is reproduced.

Neural networks are inspired by (and named after) the micro-structure of the animal brain. Units are abstractions of *neurons*, weighted connections of *synapses*, and activations represent the *firing rates* or *firing probabilities* of neurons. However, it should be emphasised that they <u>are</u> simply abstractions, and there are many additional information-processing possibilities in nature, such as more diffuse effects of hormones or peptide transmissions for example.

The weight changes in "artificial" nets are often based on a naturally occurring phenomenon discovered by Hebb in 1949. Hebb's observation was that conjoint activity in both the pre- and post-synaptic cell normally results in an increase in efficacy of the synapse in the future. This method of learning is completely *local* to the synapse, in that it is a function of only two activity levels, that impinge directly on the synapse.

Neural networks can again be compared to conventional computers. The important difference is that it is possible, in principle, for nets to operate in parallel, when each unit can be treated as a separate computational device, updating its activation independently in time. Conventional computers rely on a single, serial, Central-Processing Unit, operating to an internal clock. (In practice of course, neural nets are usually "simulated" on serial computers). Related to this is the notion that it is the *global* state of a net that is important, and thus individual disruptions to units, *e.g.* their malfunctioning, do not necessarily invalidate the useful computation of the whole system: nets are more *robust* computational devices (or have more redundancy !).

1.2.3 Short-term Memories

As mentioned earlier, one important difference between artificial and "natural" neural networks is that the latter, as far as is known, can continue to learn new information indefinitely.

This ability is particularly evident in our everyday life, since people are continually storing information for immediate, temporary purposes, which is then totally forgotten a short-while afterwards. (All too apparent when trying to remember a shopping list or telephone number !) For such reasons, people are usually posited as having a functionally separate *short-term memory.*[1]

Two quantities particularly relevant to a short-term memory are its *span* and *serial order curve.*

Spans

The main defining parameter of a short-term memory is the average number of associations that can be accurately retrieved (to within a particular criterion). This number is called the *span* of the short-term memory.

[1]Indeed, in information-processing models of human cognition, there is nearly always a component somewhere that stores information temporarily, *e.g.* in spoken language comprehension, such a component is often deemed important for backtracking during syntactic parsing of ambiguous sentences.

Serial Order Curves

Another common performance characteristic of short-term memories is their *serial order curve.* These curves usually represent the average chance of retrieving an item against its serial position in a list of items - *i.e.* in continuous learning; the time since it was last learnt.

Given a span of X items, an ideal serial order curve might be an approximate step function, where the probability of accurate retrieval is excellent until X new associations have been trained, whereupon it drops suddenly. This might correspond to a store that could hold X items perfectly, and where the learning of a new item simply entails replacing the oldest item. In people and animals however, curves always show a smoother forgetting of items over time.

1.2.4 Palimpsests

Most neural networks studied up until now cannot function as short-term memories. Eventually they reach a point when trying to learn new information leads to disasterous consequences: the retrieval of all information suddenly becomes impossible. This is called *catastrophic failure.* However, designing some method of forgetting to accompany learning, or as a natural consequence of other constraints in the net, is no trivial matter, especially since the memories are distributed over many connections.

There have been some recent suggestions in the literature. In much of this, the term *palimpsest* has been adopted to describe nets with some forgetting strategy.[2] Hence the suggestions will be referred to as *palimpsest schemes.* In a palimpsest

[2]The word "palimpsest" comes from the name of a Medieval tablet upon which text was repeatedly enscribed through time, resulting in only the most recent writing being legible, with scraps of older text being glimpsed. (Historians then have ample opportunity to squabble over interpretations of the older writings !) The analogy is really to emphasise the role of interference from more recent "memories", gradually obscuring older ones.

scheme employing both learning and forgetting processes, the two will be subsumed under the general process of *training*.

Under continuous training, the capacity of net palimpsests stabilises, rather than collapsing, resulting in a finite span and gradual forgetting over time (as evidenced in their serial order curves). Note that when nets are trained from a *Tabula Rasa* situation, where all weights are initially zero, there is normally a transient effect before the weights arrive at their asymptotic distribution (which characterises the net's stability).

1.3 Aims

The orginal aims of this project were six-fold:

1. To conduct an extensive search of the literature for references to short-term neural net memories,

2. To generalise and categorise the suggestions discovered,

3. To select as many of the suggestions as possible for implementation,

4. To compare the performances of the implementations,

5. To analyse performance mathematically, and

6. To briefly consider the suggestions from psychological, physiological and implementational view-points.

All of the above aims have been accomplished. However, the results are presented here in a somewhat different order, as described below.

1.4 Overview of Report

Chapter 2 gives a formal summary of different classes of neural network models. The two specific models considered here are then introduced: the Willshaw Net in Chapter 3, and the Hopfield Net in Chapter 4.

The literature search revealed four main palimpsest schemes (Aim 1), which are discussed, together with two further ideas for the Willshaw Net and one for the Hopfield Net, in Chapters 5 and 6. In most cases, their performance is analysed mathematically (Aim 5).

Two *simulators* were written in C for each basic model (Aim 3). The different palimpsest schemes can be selected from within the simulators (in addition to the standard, non-palimpsest net operation). Results from simulations are included in the "Performance" sections of the relevant chapters. The simulators themselves are described in the Appendices.

The different suggestions are brought together (Aim 2) and performances compared (Aim 4) in Chapter 7, which also attempts a cross-model comparison in terms of information capacity.

Chapter 8 contains some theoretical discussion on the generalised methods from the different viewpoints in Aim 6. Chapter 9 concludes and discusses possible extensions to the project.

Appendix A contains a Glossary for the notation and all the variables used in the report.

Appendix B introduces the application of signal-to-noise ratios, particularly in relation to the Willshaw Net.

Appendix C contains a User Manual for the Willshaw Net Simulator; Appendix D for the Hopfield Net Simulator, and Appendix E gives some consideration to the parameters used in simulation runs relevant to this report.

Chapter 2

Neural Network Models

This chapter gives an overview of different types of neural network models, introducing some variables and terminology used throughout the report.

2.1 Formal Description

The common features of the neural networks considered here are:

- A set of *units*, with L subsets or *layers*, each having N_l units, $l = 1, 2...L$.
- An *activation* for each unit: $\{a_i^{(l)} \in \Re \mid i = 1, 2...N_l\}$. The activation in the lth layer can be represented by the vector $\mathbf{a}^{(l)}$.
- A set of connections to a unit i in layer l from unit j in layer l', or *weights*: $\{w_{ij}^{(l,l')} \in \Re \mid i = 1, 2...N_l, j = 1, 2...N_{l'}\}$. Weights may be grouped into matrices $\mathbf{W}^{(l,l')}$ connecting layers l, l'.
- A set of P *pattern* vectors $\mathbf{v}^{(p,l)}$ for each layer l, with components: $\{v_i^{(p,l)} \in \Re \mid p = 1, 2...P, i = 1, 2...N_l\}$. (This set can be thought as having countably infinite members if patterns can be generated indefinitely.)
- The *presentation* of pattern p on layer l; "perfect", when $\forall i \ a_i^{(l)} = v_i^{(p,l)}$, or in the presence of "noise", $\mathbf{n}^{(l)}$, when $\mathbf{a}^{(l)} = \mathbf{v}^{(p,l)} + \mathbf{n}^{(l)}$, and typically $\mathbf{v}^{(l)}.\mathbf{n}^{(l)}$ is close to 1.

- An *update* rule for units, whose activations change as a function, f, of the *weighted sum* from all connected units: $a_i^{(l)} = f(\sum_{k=1}^{L} \sum_{j=1}^{N_k} w_{ij}^{(l,k)} a_j^{(k)})$.

- A *learning* rule for weights, whose values change as a function, g, of only two variables *local* to the connection: $w_{ij}^{(l,l')} = g(a_i^{(l)}, a_j^{(l')})$.

- Unit activations change over a time-course of *updating* episodes.

- Weight sizes change over an (independent) time-course of *training* episodes.

2.2 Classes of Model

Further features can be used to distinguish various *classes* of model. The following logically independent features (though correlated in practice and not necessarily conventional) can be used to classify most neural nets along several dimensions:

Partially/Fully Connected A *net* is said to be *fully connected* when every unit is connected to every other unit. If this is not the case, then it is *partially connected. Layers* can similarly said to be fully or partially inter-connected, and fully or partially intra-connected (when $l = l'$ in $\mathbf{W}$ above). Partially connected layers may still be represented by the matrix $\mathbf{W}$, simply with some of the elements being set to 0 permanently.

Iterative/Parallel Updating In one updating episode, a single unit can be updated separately (*iterative* updating) or all units in a layer can be updated together, in *parallel.*

Feedforward/Recurrent A net is said to be *feedforward* if the graph of connections has no loops; otherwise the net is *recurrent.* The term "feedforward" is adopted because the weights connecting one layer to the next often have a direction associated with them; *e.g.* with two-layer feedforward nets, one layer is often called an *input* layer, the other, an *output* layer, and the activation of output units is determined solely by the weighted activation of input units, "fed" through the weights. (The activation of the input units is provided by presentation of an input pattern).

Symmetric/Asymmetric Weights If a net's weights are such that $\forall i \forall j \; w_{ij} = w_{ji}$, then that net has symmetric weights. (This may depend on the learning function.)

Continuous/Discrete Activations Whether $a_i^{(l)}$ can vary continuously or only take one of a finite set of values.

Continuous/Discrete Weights Whether $w_{ij}^{(l,l')}$ can vary continuously or only take one of a finite set of values.

Linear/Nonlinear Activation Functions Classification based on whether f is linear. In most interesting cases, it is non-linear and dependent on a unit-based parameter called the *threshold*, τ_i.

Linear/Nonlinear Learning Functions Classification based on whether g is linear.

Instantiation of variables L, f and g inside a class yields particular *models*. (Sometimes the term *architectures* is used to distinguish different values of L and particular connectivities.) Instantiation of N_l yields an individual *net*. One final important definition for the operation of an individual net is:

Auto-Association/Hetero-Association When two or more different patterns are associated by a net, it is operating under *hetero-association*. When a net is only learning one pattern (associating its components with "each other"), it is functioning as an *auto-associator*. Though any class of model can in principle operate under either condition, the conditions do relate to particular architectures since the notion of a layer is usually to distinguish which patterns are being associated.

2.3 Exploration of Models

Complete exploration of this space of different models, together with different methods of introducing forgetting, would be extremely arduous. Rather, this project initially concentrates on two particular, standard models: the Willshaw Net (WN) and the Hopfield Net (HN). These particular models stand out like mountains in the space of different neural networks; from the top of which, most authors in the literature have also started their journey.

The WN is a fully inter-connected, feedforward net with 2 layers and parallel update. Activations and weights are both discrete and binary, whilst both activation and learning functions are Heaviside functions.

The HN is a fully intra-connected, recurrent net of 1 layer, generally with symmetric weights (since it is normally used for auto-association). Activations are binary and updated iteratively, but weights are continuous. The activation function is obviously non-linear, whilst the learning function is linear.

Within these models, the investigation has mainly been to discover and optimise relations between variables in the learning, forgetting and activation functions. The net architectures are fixed, as is the size of the nets, although there is some interest in how the properties of the nets scale with N_l.

Chapter 3

Willshaw Net

This chapter gives an introduction to the standard operation of the Willshaw Net, and considers its capacity as a memory device under noise-free memory cueing.

3.1 Introduction

The WN is a fully inter-connected, feed-forward associative memory, with one layer of *input* units, one layer of *output* units and binary-valued weights, or *switches.* The net associates a number of *pattern pairs*, each input and output pattern in the pair being represented as a vector with binary-valued components. When these patterns presented to the net, an *active* unit then corresponds to a pattern component of value 1.

Learning of these associations is by simple Hebbian reinforcement, where the switch connecting input unit i and output unit j is turned on, or *triggered*, whenever there is conjoint activity of both units.

An association has been *stored* when there is good *retrieval* of the output pattern. Good retrieval is interpreted as a (near-)faithful reproduction of the output pattern over the output units, given a pattern of activity over the input units similar to the corresponding input pattern (a *cue*). Reproduction is via parallel update and thresholding of the weighted sum for each output unit.

3.2 Mathematical Characterisation

Let a WN have N_I input units and N_O output units. The weight matrix $\mathbf{W}$ then has $N_I \times N_O$ elements, with a weight $w_{ij} = 1$ corresponding to a triggered switch.

Consider input and output pattern vectors $\mathbf{v}^{(I)}$ and $\mathbf{v}^{(O)}$, where $v_i^{(I)}, v_i^{(O)} \in \{0, 1\}$. To learn this new pattern pair, the WN learning rule is:

$$w_{ij} \to g(w_{ij} + v_i^{(O)} v_j^{(I)}) \tag{3.1}$$

where the learning function g is:

$$g(x) = \begin{cases} 1 & \text{if } x \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

Let the input unit activities be represented by vector $\mathbf{a}^{(I)}$ and output unit activities by $\mathbf{a}^{(O)}$. Then the updating rule is then:

$$a_i^{(O)} \to f(\sum_{j=1}^{N_I} w_{ij} a_j^{(I)}) \tag{3.2}$$

where the activation function f is a non-linear thresholding function:

$$f(x) = \begin{cases} 1 & \text{if } x \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

As apparent, the threshold is common to all output units.

Let the number of components of value 1 in pattern p be M_p; the ratio $F_p = \frac{M_p}{N_p}$ is referred to as the *pattern coding.* For simplicity, let M_p be constant for all input patterns at M_I, and for all output patterns, at M_O. Then, from consideration of the learning rule, $\tau = M_I$ is the relevant threshold setting.

The quality of retrieval can be measured by the *hamming distance* index. The hamming distance between two N-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$, H($\mathbf{x}$, $\mathbf{y}$), is defined as the number of components by which the two vectors differ:

$$\begin{aligned} H(\mathbf{x}, \mathbf{y}) &= \sum_{i=1}^{N} h(x_i, y_i) \qquad (3.3) \\ h(x, y) &= \begin{cases} 0 & \text{if x = y} \\ 1 & \text{otherwise} \end{cases} \end{aligned}$$

The hamming distance between $\mathbf{a}^{(O)}$ and $\mathbf{v}^{(O)}$ is often referred to as the output hamming distance, H_O. An $H_O = 0$ means identical vectors and perfect retrieval.

In practice, retrieval is rarely perfect. There exist two possible types of error in the net output:

1. *Spurious Errors* $\quad a_i^{(O)} = 1,\ v_i^{(O)} = 0$
2. *Omission Errors* $\quad a_i^{(O)} = 0,\ v_i^{(O)} = 1$

In standard, noise-free operation of the WN with constant pattern coding, only spurious (or "commission") errors are possible, since switches can only ever be turned on. Indeed, the mean number of spurious errors increases as more pattern pairs are learnt. With palimpsest schemes however, where switches can be turned off, both types of error can occur.

3.3 Theoretical Capacity

3.3.1 Hetero-association

[Willshaw *et al* 69] derive conditions for efficient use of the WN under hetero-association, by attempting to keep the number of spurious errors in retrieved patterns small.

Consider the learning of R random pattern pairs. The probability that a particular switch, w_{ij}, has been triggered at some time during the training of these patterns, p, is:

$$p \;=\; 1 - (1 - F_I F_O)^R \simeq 1 - \exp\left(F_I F_O R\right) \tag{3.4}$$

given small values of F_I and F_O, *i.e. sparse* pattern coding. Rearranging this equation allows an expression for R:

$$R \simeq -\frac{N_I N_O}{M_I M_O} \ln\left(1 - p\right) \tag{3.5}$$

The probability p is sometimes also viewed as the *loading density* of a net. Note that as $p \to 1$, the WN fails as a memory device, since, given any input pattern, all output units will be activated.

To keep the number of spurious errors small, a criterion can be chosen, *e.g.* one expected spurious error per net output, *i.e.* $E[H_O] = 1$.

Given that the pattern pairs are randomly chosen with a constant M_I and M_O, the expected number of spurious errors is $(N_O - M_O)\, p^{M_I}$ (though this is making some assumptions - see next section). Then a slightly more stringent limit of good performance can be set by the condition:

$$N_O\, p^{M_I} = 1 \tag{3.6}$$

From Information Theory considerations [Willshaw 71] the information efficiency of a net, ξ, can be defined as:

$$\xi = \frac{R\, log_2\,(C^{N_O}_{M_O})}{N_I N_O} \tag{3.7}$$

where C^n_r is the number of combinations of r items in n.

By substituting in expressions for R and M_I from Eqs 3.5 and 3.6, ξ can be determined for the maximum number of pattern pairs stored and retrieved with an "average" of one spurious error, R_c. R_c is sometimes termed the *capacity* of a net. Using Stirling's approximation, ξ then conveniently simplifies to a function of p only:

$$\xi \simeq log_2(p) ln(1-p) \tag{3.8}$$

Maximising ξ with respect to p gives maximum information efficiency (of $ln2$) in the large N_O limit at $p = p_c = 0.5$ and $M_I = log_2(N_O)$, *i.e.* very sparse coding. In this case, when $N_I \simeq N_O = N$:

$$R_c = O\Big[\ (\tfrac{N}{log(N)})^2\ \Big] \tag{3.9}$$

3.3.2 Auto-association

When the WN is used for auto-association, $\forall$p$(\mathbf{v}^{(I,p)} = \mathbf{v}^{(O,p)})$, $M_I = M_O = M$, $N_I = N_O = N$, and $\mathbf{W}$ is a symmetric matrix. Note however that optimum performance with perfect cueing could actually be achieved when $\mathbf{W} = \mathbf{I}$ and $\tau = 1$. It is only really the possibility of *noise* in the input patterns that makes auto-association interesting.

However, the above analysis of capacity is not strictly valid for the auto-associative case. Here, the value for p in Eq 3.4 is a poor approximation when M is small. Considering a particular output line j, the probability of the weight on the leading diagonal, *i.e.* w_{jj}, being triggered in R patterns, p_1, is greater than the probability for off-diagonal weights, p_2:

$$p_1 = 1 - \left(1 - \tfrac{M}{N} \right)^R \tag{3.10}$$

$$p_2 = 1 - \left(1 - \tfrac{(M-1)M}{(N-1)N} \right)^R \tag{3.11}$$

The probabilities p_1 and p_2 are linked by the equation:

$$N^2 p = N p_1 + N(N-1) p_2 \tag{3.12}$$

and since the second term dominates for large N, an approximate value for R under sparse-coded auto-association is:

$$R \simeq -\frac{N^2}{M(M-1)} \ln(1-p) \tag{3.13}$$

The amount of information stored in an auto-associative net can also be a misleading quantity, since that information can only be retrieved given a proportion of that same information in the first place ! (Though, by defining an *information storage* capacity for nets, [Palm 92] shows auto-association is half as efficient as hetero-association under this measure). This caveat aside, the capacity of an auto-associative WN under optimal coding can similarly be said to increase with the square of $\frac{N}{log(N)}$.

3.4 Unit Usage

The above analysis is only a first approximation to the maximum number of pattern pairs usefully stored. Results from simulations give R_c significantly smaller than the theoretical prediction. This discrepancy is because Eq 3.6 contains an assumption now known as the *unit usage* approximation. This is that the proportion of triggered switches connected to each output unit is the same for all output units. However, with randomly generated patterns, the unit usage follows a binomial distribution with characteristic probability F_O.

This means that the distribution of weighted sums on output units that should be inactive is not symmetrical, but negatively skewed. This results in a slightly greater chance of output units producing a spurious error than predicted. (Moreover, under auto-association, the symmetry of the weight matrix also introduces correlations amongst the distribution of sums.)

Generally, the discrepancy is smaller for sparse pattern codings (though output errors also increase as $M_I \to 0$). The discrepancy is actually minimised for the optimum encoding above, of $M_I = log_2(N_O)$. However, analysis of the general case is very complex. In all subsequent analyses, the unit usage approximation has been made for mathematical tractability. For further details, see [Buckingham & Willshaw 92].

3.5 Performance

Results from the Willshaw Net Simulator are consistent with the above analysis. The Figs 3–1 to 3–4 come from a 512(9) *square net, where* $N_I = N_O = 512$ *and* $M_I = M_O = 9$*, operating under hetero-association, which has a theoretical maximum capacity* $R_c \simeq 2243$.

Fig 3–1 shows the average H_O for pattern pairs 1 to R, plotted against R, where R is the number of pattern pairs trained (from a Tabula Rasa). When $\overline{H_O} = 1$, it can be seen that $R \simeq 2100$. The discrepancy between 2100 and 2243 is due to the large N approximations and particularly the unit usage assumption made in calculating R_c. Fig 3–2 is a graph of unit usage for each output unit when $p = p_c$, which gives an indication of the variability of this quantity (mean is 256.1 with standard deviation 27.6).

Fig 3–3 shows the loading density, p, as a function of R. Here, as expected, $p = 0.5$ at $R \simeq 2243$. As more pattern pairs are trained beyond this point, p asymptotically approaches 1 (when all switches in the net are triggered).

Fig 3–4 is a plot of the number of patterns with $H_O \leq 1$, S, against R. S corresponds to the number of output patterns *reliably retrieved.* Also shown is a plot of S/R against R. It shows clearly the *catastrophic failure* characteristic of

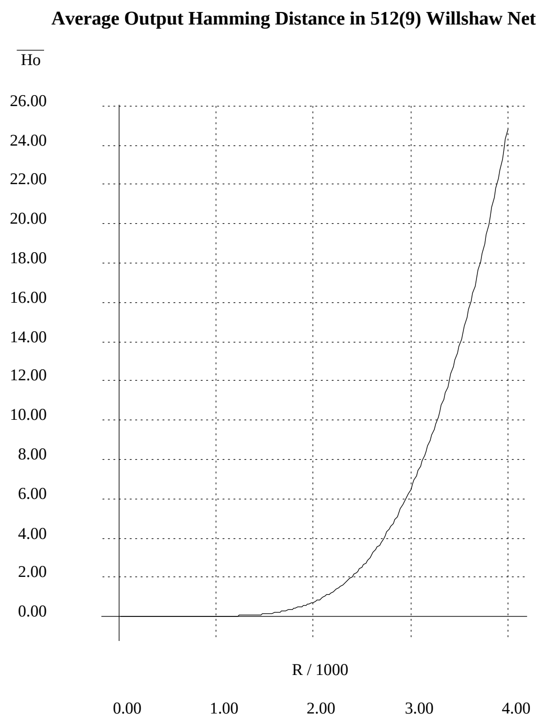


Figure 3–1: Average Output Hamming Distance in Standard Willshaw Net

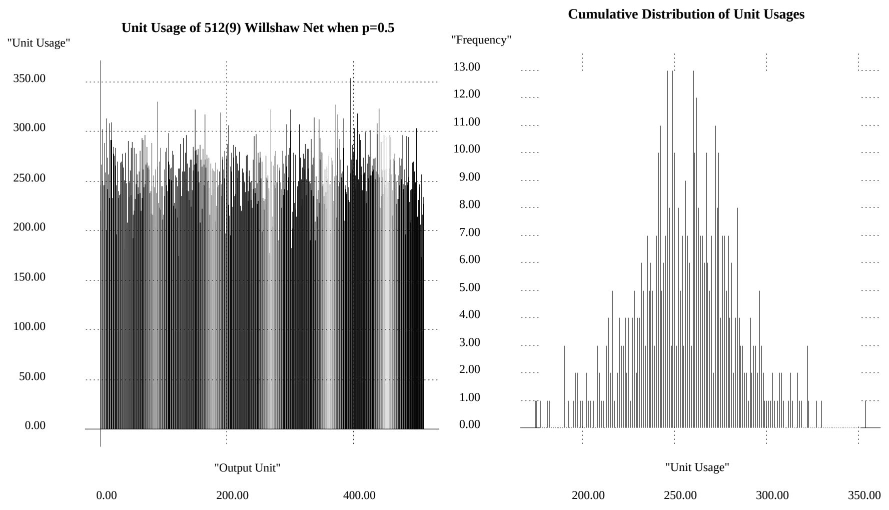


Figure 3–2: Unit Usage in Willshaw Net when $p = p_c$

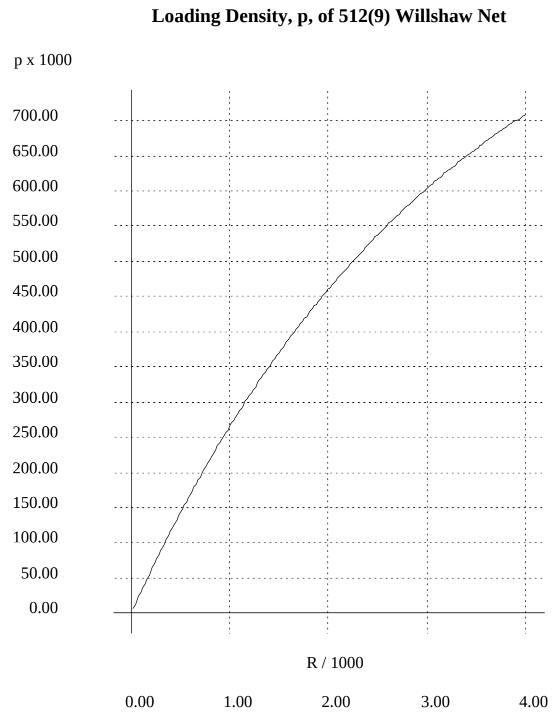


Figure 3–3: Loading Density in standard Willshaw Net

the WN under excess loading: by $R \simeq 3200$, the net has ceased to function as an associative memory.

When $p = p_c$, $S \simeq 1370$. Obviously, this value is not the same as R_c since, although $\overline{H_O} = 1$, there will be many net outputs with an H_O of 2,3 or more. Each pattern has a finite chance of *not* being retrieved reliably; this chance increases slowly to 0.5 when $R \simeq 2243$ - after that it increases rapidly.

The maximum value of S is $\simeq$ 1670, when $R \simeq 1900$. Thus operation at "maximum information efficiency", in the sense defined in the previous section, does *not* give the greatest number of output patterns reliably retrieved. Rather, this occurs at sub-optimal loadings, $p \simeq 0.43 < p_c$.

Simulation results when this net is operating under auto-association give $p = p_c$ *when* $R \simeq 2500$. *This is in agreement with the prediction from Eq 3.13. This in turn produces a larger maximum value for* $S \simeq 1920$.

3.5.1 A word on spans

The definition of the parameter S above is that of the *span* of a net. However, since the net here is not functioning as a palimpsest, it is not really appropriate to say it has a maximum span of about $1,700$ associations.

Note that S is a new measure of net performance, which has not been utilised before in the analysis of the WN. It would seem a better index of performance of a palimpsest than its "capacity", since the interest is in near-perfect retrieval of the most recently learnt associations. Near-perfect, or "reliable" retrieval requires $H_O < H_L$, where H_L is the *hamming limit*, chosen as 2 here.

It is useful to consider the relation between S and R. Their maximum values occur at different loadings because, whereas R is only a logarithmic function of p, S is a multiplicative function of both this logarithm and another function that decreases with p. Since patterns with $H_O > 1$ do not contribute to S, an estimate of the expected value of S is given by:

$$S = -\frac{N^2}{M^2} ln(1-p)Q(p) \tag{3.14}$$

where $Q(p)$ is the probability of obtaining 0 or 1 spurious errors:[1]

$$Q(p) = (1-p^M)^{N-M} + (N-M)(1-p^M)^{N-M-1}p^M \tag{3.15}$$

Thus S would not be expected to have its maximum value at the same loading that gives "capacity-optimal" performance when $R = R_c$.

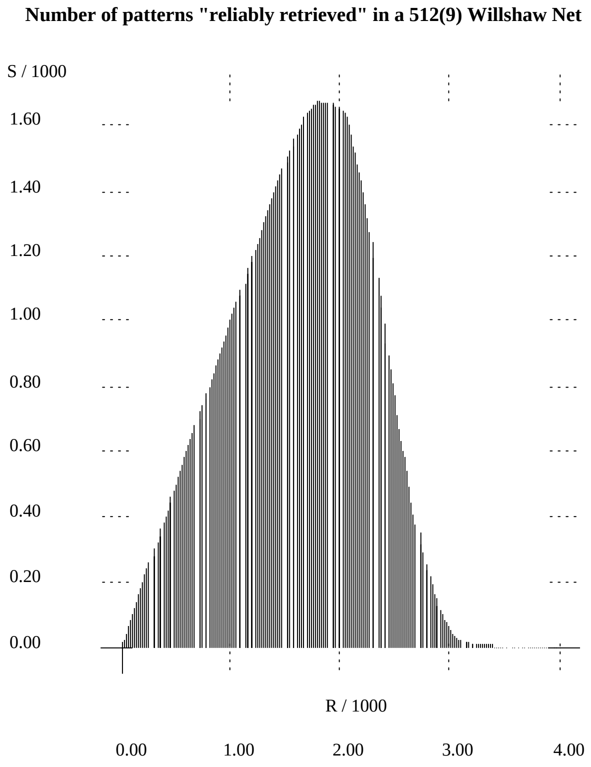
Alternatively, the two quantities, S and R can be simply related when $R = R_L$, where R_L is such that $E[H_O] = H_L = 2$: then $S \simeq \frac{3}{e^2} R_L$ (substituting $(N-M)p^M = 2$ into the above equation). This can be checked from Fig 3–1, where $R_L \simeq 2400$, and from Fig 3–4, where $S(R = R_L) \simeq 970$.

Note finally that the WN simulation results in this report come from a square net with $N = 512$ and capacity-optimal $M = log_2(N) = 9$. The choice of N comes from a play off between the desire for large N and the desire for reasonable

[1]This is a special case of Eq 5.22.

amounts of computer-time. The second constraint is reinforced by the fact that R needs to be large for an accurate estimation of span ($\simeq 10,000$).

Since maximum S does not occur when $p = p_c$, this suggests that the "span-optimal" pattern coding may not be $F = \frac{log_2(N)}{N}$ (to actually determine the span-optimal coding would seem to be very complicated). However, it is still the case that optimum spans will come with sparse coding - as later chapters show - so M, although no longer exactly selected by theory, has been kept as 9. Indeed, since M and N are only theoretical variables, not an experimental ones (fixed for most simulations), the choice is not immediately important.



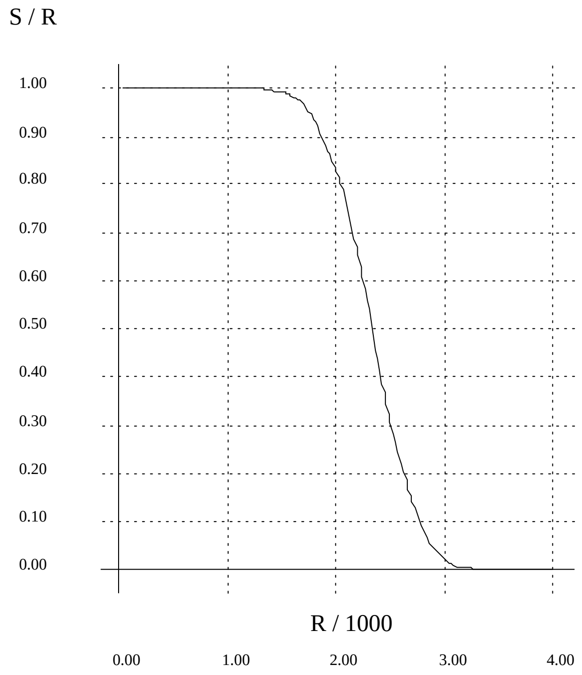


Figure 3–4: Number of Patterns Reliably Retrieved in Willshaw Net

Chapter 4

Hopfield Net

This chapter gives an introduction to the standard operation of the Hopfield Net, and considers its capacity as a memory device under noise-free memory cueing.

4.1 Introduction

The HN is a fully intra-connected, recurrent associative memory, with a single set of *units* connected by real-valued weights. The net can store a number of patterns, represented as vectors with *spin*-valued components of 1 or -1.[1] The HN is normally used for autoassociation. (Although it is sometimes used to store hetero-associative sequences of patterns. [Parga 89] gives a good overview of some popular modifications to the basic HN.)

When these patterns are imposed on the units, an *active* unit then corresponds to a pattern component of 1; an *inactive* unit, the value of -1. The activation vector of a net at any one time is referred to as the *state* of that net.

Learning is by a variation of Hebbian reinforcement, where the weight connecting units i and j is strengthened whenever there is either conjoint activity or

[1] *Spin* values, rather than the binary values used in the WN, are employed for historical and conventional reasons. They also make mathematical characterisation of the HN more slightly more concise. Note that the two conventions are related by a trivial linear transformation.

conjoint inactivity of both units. Whenever one unit is active but the other is inactive, the weight is weakened.

Retrieval of patterns is an iterative process, where units update their activation one at a time (asynchronously). A pattern is said to have been *stored* when, starting at a certain state, the net *relaxes* after several updates to a *stable* state similar to that pattern.

4.2 Mathematical Characterisation

Let a HN have N units and a weight matrix, $\mathbf{W}$. Consider a pattern vector $\mathbf{v}$, where $v_i \in \{-1, 1\}$. Let the value of each component be chosen at random with equal probability (such vectors are called *uniform signary* vectors); then $\forall p\ E[F_p] = 0.5$.

The HN Learning Rule is:

$$w_{ij} \rightarrow w_{ij} + \eta v_i v_j \tag{4.1}$$

where η is a scaling constant, often inversely proportional to N (*i.e.* the learning function g is linear over all patterns).

Let the state of the net be represented by vector $\mathbf{s}$ ($= \mathbf{a}^{(1)}$). Then the Update Rule is:

$$s_i \rightarrow f(\sum_{j=1}^{N} w_{ij} s_j) \tag{4.2}$$

where f is the thresholding activation function:

$$f(x_i) = \begin{cases} 1 & \text{if } x_i > \tau_i \\ -1 & \text{if } x_i < \tau_i \end{cases}$$

For simplicity, let $\forall i(\tau_i = 0)$. The N weighted sums are often said to comprise the "local field", $\mathbf{h}(\mathbf{s})$. In the case that $h_i = 0$, the correct update is $s_i \rightarrow s_i$.

View the update dynamics as over iterations t. Units are said to be stable when their current sign matches that of the local field (then $s_i(t+1) = s_i(t)$). Units are chosen at random to be updated, until each unit is stable. Call this stable state of the net $\hat{\mathbf{s}}$. Thus relaxation is a non-deterministic process with fixed end-points.

Provided weights are symmetrical ($w_{ij} = w_{ji}$) and there are no self-connecting weights ($w_{ii} = 0$), there exists a Lyapunov function associated with the update dynamics, that defines a quantity often termed the *energy*, E, of a state. The utility of this quantity was first recognised by [Hopfield 82]:

$$E(\mathbf{s}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} s_i s_j \tag{4.3}$$

Under these constraints, the quantity E can be shown never to increase: each update will either lower the energy, or leave it unchanged.

In energy terms, the stored patterns of a net correspond to local minima of an energy "landscape" in N-dimensional space. The relaxation of a net is then viewed as a trajectory through this space, from the point representing the starting state, to the nearest minimum, which corresponds to a final, stable state.[2]

The stored patterns are often called *attractors* because of this "basin of attraction" for nearby starting states. However, not all patterns learnt are stored - *i.e.* produce perfect minima - and they are rarely the only attractors in the net. This is because the linear superposition of patterns in training also creates *spurious* attractors. A net stabilising in a spurious attractor would not correspond to retrieval of any learnt pattern.

The quality of retrieval can again be measured by the hamming distance index, H, (see Eq 3.3) between the vectors $\hat{\mathbf{s}}$ and target pattern $\mathbf{v}$. There is an alternative measure of retrieval quality that is common in the literature. This is the *overlap* (or inner product), $m(\mathbf{x}, \mathbf{y})$ defined as:

$$m(\mathbf{x}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N} x_i y_i \tag{4.4}$$

[2]In practice, the constraints of symmetric and non-self-connecting weights can be relaxed and the net can still function adequately - although it may not always reach a stable state. However, there is a lemma associated with the updating algorithm: if a net has not reached a stable state in N updates, then it will not reach a stable state at all (it will oscillate indefinitely).

The net output overlap is then $m_o = m(\hat{\mathbf{s}}, \mathbf{v})$. Perfect retrieval then gives $m_o = 1$, whilst the case of $m_o = -1$ corresponds to the stable state being the inverse of the target pattern.[3] (This is not uncommon, since, due to the symmetrical nature of the learning rule, the inverses of trained patterns also become attractors in the energy landscape.)

Of course, overlap and hamming distance are simply related for spin vectors by $m = 1 - \frac{2H}{N}$. However, though m has the advantage that gives a good indication of similarity, irrespective of N, H will occasionally be used for comparison with the WN.

4.3 Theoretical Capacity

4.3.1 Physical Analogies

If patterns are orthogonal, they can be superposed without any interaction or creation of spurious attractors.[4] In practice, the "pseudo-orthogonality" of random patterns [Hopfield 82] will be respected when the number of patterns trained, R, is small compared to N. As R grows, the likelihood of "cross-talk" between the patterns increases. This extra noise soon causes instability of learnt patterns, which eventually determines the capacity of the net.

As it happens, physicists have been studying similar interactions amongst the components of complex systems in nature (like the interaction of magnetic domains in some materials). The standard HN is actually a special case of a more general class of models where the update rule itself is not deterministic, but stochastic. The probability of changing the activation of a unit is a function of both the local field and a parameter referred to as the "temperature" of the net [Hertz *et al* 91]. Nets operating under non-zero temperature are called "Boltzmann Machines".

[3]If a net does not stabilise in N updates, a default $m_o = 0$ is assigned

[4]Unless N such mutually-orthogonal patterns are learnt (*i.e.* the patterns comprise an orthogonal basis for the N-dimensional space), in which case $\mathbf{W}$, with diagonal terms, coverges to $N\mathbf{I}$.

Physicists have developed powerful methods for exposing phase-transition diagrams of such systems, parameterised by the temperature and the *loading* of the net. The loading of the net, α, is simply the ratio of the number of patterns trained, R, against the net size N:

$$\alpha = \frac{R}{N} \tag{4.5}$$

When the temperature is 0, the standard HN is recovered, and, there is a critical capacity, α_c, beyond which good retrieval of patterns suddenly becomes unlikely [Amit *et al* 85]. Theoretical and numerical results give $\alpha_c \simeq 0.138$, whence m_o drops from about 0.97 to 0.35.

Thus, given a reliable retrieval criterion, $m_L = 0.97$ (say), the maximum number of patterns that can be reliably retrieved, just before catastrophic failure occurs, is $O[N]$, *i.e.* linear in N.[5]

4.3.2 Analysis of Local Fields

Irrespective of the powerful methods used above, it is useful to give simple consideration to how a HN fails under non-palimpsest conditions. This involves making two assumptions.

Firstly, the capacity of a net can be approximated by the number of learnt patterns that correspond to *initially* stable states. If a presented pattern has one or more of the units initially unstable, then, as borne out in practice, it is very likely that the state will settle into a quite different attractor (a "cascade" effect) - such a pattern cannot be counted as a stored "memory".

Secondly, the N^2 weights are treated as random, independent variables. In reality of course, there are correlations between these variables, *i.e.* $w_{ij} = w_{ji}$. (Such correlations can only justifiably be ignored in partially-connected, asymmetric nets, where connections between units are very sparse. Then the dynamics can be solved exactly [Derrida *et al* 87].)

[5]There do exist "multi-connected models" which have interactions from more than $n = 2$ units and have Hamiltonian (Lyapunov) functions of order n. Their capacities generally increase with N^{n-1} [Horn & Usher 88].

Now the necessary and sufficient condition for a pattern, $\mathbf{v}$, to be a stable state of the net is that, for all i, the sign of v_i is the same as the sign of h_i. This is equivalent to the requirement that [Personnaz *et al* 86]:

$$\mathbf{W}\mathbf{v} = \mathbf{A}\mathbf{v} \tag{4.6}$$

where $\mathbf{A}$ is some diagonal matrix with non-negative elements - in other words:

$$\forall i \sum_{j=1}^{N} w_{ij} v_j v_i > 0 \tag{4.7}$$

Now consider the mean and variance of the local fields for the pth pattern, $\mathbf{v}^{(p)}$ after R have been learnt [Gordon 87]. Let $\eta = \frac{1}{N}$; then the average local field over all learnt patterns for unit i is given by:

$$\overline{h_i(\mathbf{v}^{(p)})} = \frac{1}{N} \sum_{j=1}^{N'} \sum_{r=1}^{R} \overline{v_i^{(r)} v_j^{(r)} v_j^{(p)}} = \frac{1}{N} \sum_{j=1}^{N'} (\sum_{r=1, r \neq p}^{R} \overline{v_i^{(r)} v_j^{(r)} v_j^{(p)}} + v_i^{(p)}) \tag{4.8}$$

where $\sum_j^{N'} = \sum_{j \neq i}^{N}$. Since the first term above is the sum of independent variables of value -1 or 1, the expected value is 0. Thus:

$$\overline{h_i(\mathbf{v}^{(p)})} = \frac{N-1}{N} v_i^{(p)} \simeq v_i^{(p)} \tag{4.9}$$

Therefore, the net state should - in the average - correspond to $\mathbf{v}^{(p)}$ (if the initial state is not a learnt pattern, then $\overline{h_i(\mathbf{s})} = 0$).

The second moment of the local field is:

$$\begin{aligned} \overline{h_i^2(\mathbf{v}^{(p)})} &= \frac{1}{N^2} \overline{\sum_{j=1}^{N'} \sum_{r=1}^{R} v_i^{(r)} v_j^{(r)} v_j^{(p)} \times \sum_{k=1}^{N'} \sum_{s=1}^{R} v_i^{(s)} v_k^{(s)} v_k^{(p)}} \\ &= \frac{N-1}{N^2} R + \frac{(N-1)(N-2)}{N^2} v_i^{(p)} v_i^{(p)} \end{aligned} \tag{4.10}$$

The first contribution comes from the terms $j = k$, whilst the second comes from the terms $j \neq k$, and is $\overline{h_i}^2$ (*i.e.* ignoring terms of order $\frac{1}{N}$). The variance of the local field, σ_i, is then:

$$\sigma_i = \overline{h_i^2} - \overline{h_i}^2 \simeq \frac{R}{N} \tag{4.11}$$

Thus even when the initial state corresponds to a trained pattern, when $\frac{R}{N}$ is large enough, there is some probability that Eq 4.7 will be violated. Ignoring correlations between weights, this probability, Q, can be assumed equal for all i, and can be expressed as a Gaussian function of $\frac{\sigma}{\overline{h}^2}$:

$$Q(\frac{\sigma}{\overline{h}^2}) = \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{0} \exp{(-\frac{(z - \overline{h}v^{(p)})^2}{2\sigma})} dz \tag{4.12}$$

For small x, the function $Q(x)$ vanishes like $\exp{(-x^{-2})}$ and is linear in the neighbourhood of $x = x_* = \frac{1}{3}$, the inflexion point. It can be approximated by a straight line passing through x_* of slope $\frac{dQ}{dx}|_{x=x_*}$. This crosses the abscissa at $x = x_0$, where $Q(x_0) = 0$. Then below x_0, trained patterns are expected to be initially stable states; beyond x_0, unstable units are expected.

The numeric value of x_0 is 0.153. Since $\overline{h}^2 = 1$ from Eq 4.9 and $\sigma = \frac{R}{N}$, x_0 corresponds to the capacity α_c, and is in close agreement with the more powerful methods of [Amit *et al* 85]. Thus a signal-to-noise analysis of the local fields for each unit gives a good prediction of the point at which a HN undergoes catastrophic failure. This will be utilised when the palimpsest schemes in Chapter 6 are considered.

4.4 Performance

Results from the Hopfield Net Simulator are shown in Figs 4–1 to 4–3. Here, $N = 512, \eta = 1.0$ *and the expected maximum number of patterns reliably retrieved is* $\simeq 71$.

Fig 4–1 is a plot of the number of patterns with $m_o > 0.97$ ($H_O < 7$) against the number of patterns trained, R. It shows clearly the *catastrophic failure* of the HN, since by $R \simeq 120$, the net has ceased to function as an associative memory. Fig 4–2 shows the average overlap of all patterns trained so far, which shows the failure from a different perspective.

The maximum ordinate value is in fact 62, which is somewhat short of the above prediction. This is probably because the prediction is based on the assumption $N \to \infty$, and finite N effects reduce the capacity in practice.

Fig 4–3 shows the distribution of average incident weights for each of the 512 units when $R = 300$. The mean of all weights is -0.04, *i.e.* close to 0, but the variance is very large, $\simeq 300.26$.

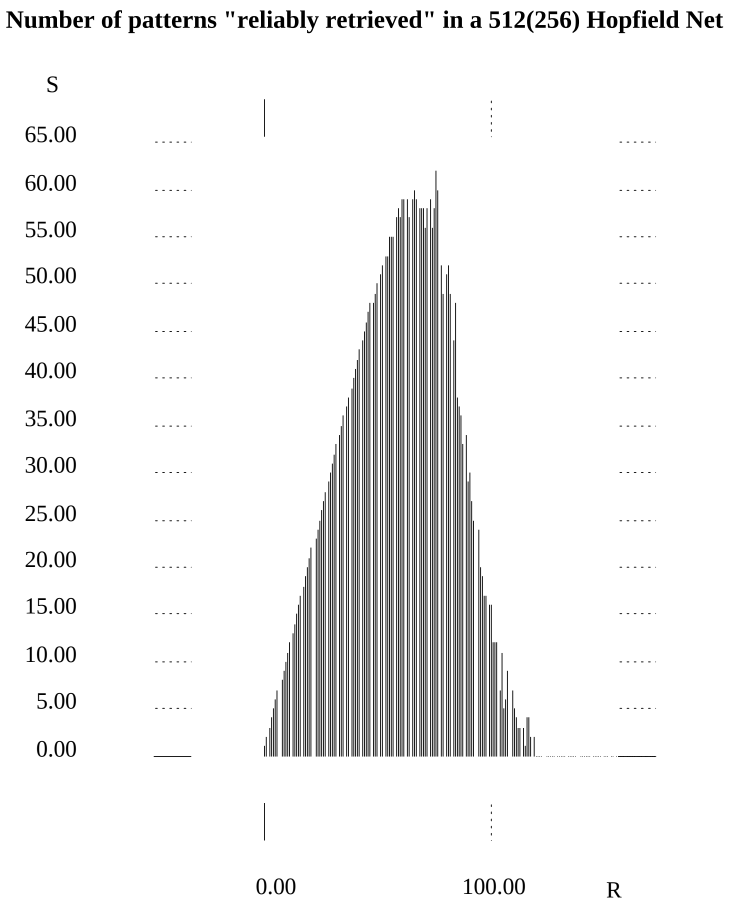


Figure 4–1: Number of Patterns Reliably Retrieved in Hopfield Net

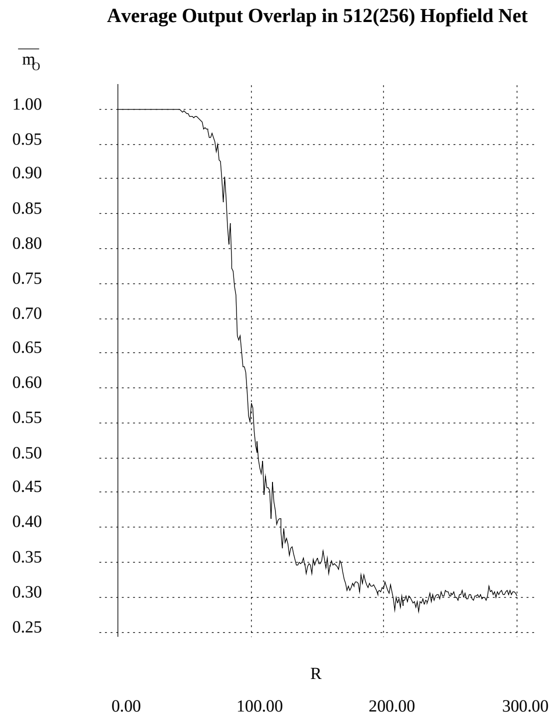


Figure 4–2: Average Overlap in standard Hopfield Net

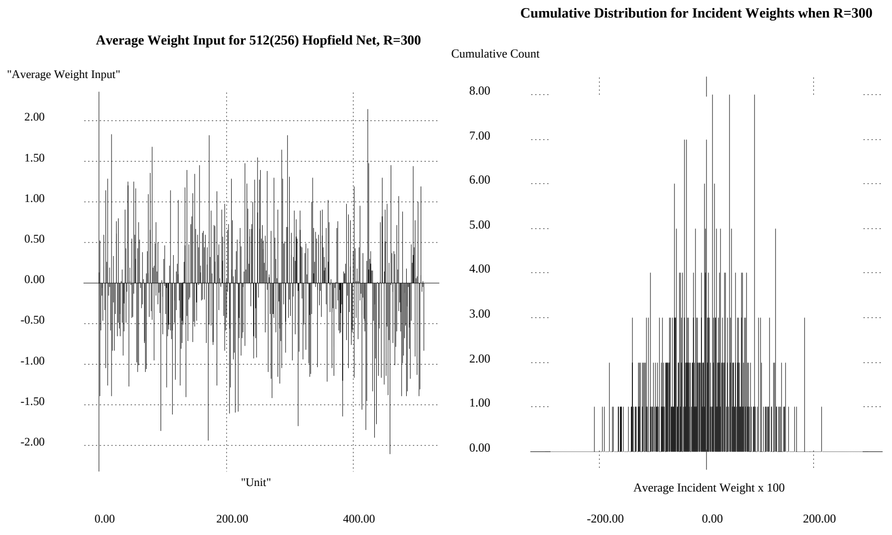


Figure 4–3: Average Incident Weights in Hopfield Net when $R = 300$

Chapter 5

Forgetting in the Willshaw Net

This chapter examines three main proposals for palimpsest schemes in the Willshaw Net: *random resetting*, *weight ageing* and *generalised learning.* Approximate predictions of the noise-free, hetero-associative spans under each method are derived, where possible, via a probabilistic analysis of switch triggering.

5.1 Random Resetting

The normal WN learning rule *always* triggers a switch given conjoint input and output unit activity. Thus, immediately after having learnt an association, testing with the same input pattern will produce a weighted sum of M_I to each target output unit, which can thus be used as the threshold to distinguish true *signals* from background *noise.*

Random Resetting involves turning off random switches with a (small) probability. An episode of forgetting precedes the learning of a new pattern pair. Since forgetting can turn off some of the switches triggered by previous patterns, there is now a chance of omission errors when those patterns are retested. Thus it may be advisable to adjust the threshold, to reduce the chance of these errors (paying by the increased chance of spurious errors). Moreover, it is worth investigating performance in palimpsest schemes when conjoint activity does not always trigger a switch - *i.e.* it is only likely that it will.

Consideration of this general case is given later. First however, a special case of Random Resetting is described where the threshold is kept normal and the net loading is capacity-optimal at $p = p_c$. Examination of this special case will clarify some important concepts, such as *survival time*.

5.1.1 Analysis of Special Case

The Method

The original suggestion of [Willshaw 71] sought to stabilise the loading density of a net at the constant level p_c. Viewing time as discretized into single training episodes, the aim is to maintain the average p constant over time at:

$$\overline{p(t)} = p_c = 0.5 \tag{5.1}$$

This requires the number of switches turned on by a pattern learnt at time t to be equal to the number of random switches turned off (*reset*). Now the former quantity is simply:

$$M_I \times M_O \times (1 - p(t)) \tag{5.2}$$

Let the probability of resetting a triggered switch be r; the latter quantity is then:

$$N_I \times N_O \times p(t) \times r \tag{5.3}$$

Equating and rearranging, we obtain an expression for r:

$$r = \frac{M_I M_O \, p(t)}{N_I N_O (1 - p(t))} \tag{5.4}$$

which, with $p(t) = p_c$, simplifies to:

$$r = \frac{M_I M_O}{N_I N_O} = F_I F_O \tag{5.5}$$

Thus one *training* episode in this Random Resetting algorithm involves learning each pattern pair normally (Eq 3.1), but preceding each learning with an iteration over approximately $p_c N_I N_O$ triggered switches, resetting each with probability r.

Survival Time

The survival time of a pattern pair association is the number of subsequent training episodes before that association is deemed "forgotten", *i.e.* when the output pattern can no longer be reliably retrieved. To simplify analysis of this quantity, consider the case of a square net where $N_I = N_O = N$, $M_I = M_O = M$ and $F = \frac{M}{N}$.

Take a particular switch, w_{ij}, which is on at time t with probability $\hat{p}(t)$. Consider the probability that w_{ij} is still on after learning a pattern pair at time $t+1$, subsequent to an intervening forgetting episode:

$$\hat{p}(t+1) = \hat{p}(t)(1-r) + (1-\hat{p}(t))F^2 \tag{5.6}$$

The first term is the probability that w_{ij} is not reset during forgetting, whilst the second is the probability that w_{ij} is triggered by the learning of the new patterns.

Substituting $q(t) = \hat{p}(t) - p_c$ and the expression for r in Eq 5.5, the following recurrence relation emerges:

$$q(t+1) \quad = \quad q(t)(1-2F^2) \tag{5.7}$$

Thus after R training episodes:

$$q(t+R) = q(t)(1-2F^2)^R \simeq q(t)(1-2F^2R) \tag{5.8}$$

providing $F^2 \ll 1$, *i.e.* coding is sparse.

It is useful to distinguish two types of triggered switches in the net. There are those that are supposed to remain on in order to store a given association. Let the probability that they do remain so be $p_s(t)$ (where "s" stands for "signal"). The second type are those not relevant to that particular association, which have been triggered by the learning of other pattern pairs, and only contribute background "noise" to that association. Let these be turned on with probability $p_n(t)$.

Now $p_s(t)$ and $p_n(t)$ are not truly independent, since they are related to the constant p_c by:

$$N^2 p_c = M^2 p_s(t) + (N^2 - M^2) p_n(t) \tag{5.9}$$

However, in the sparse coding limit, they can be effectively treated as such, and $p_n(t)$ can also be assumed constant over time, at $p_n(t) = p_c$.

Immediately after learning a particular pattern pair that triggers w_{ij} at time 0, $p_s(0) = 1$. Thus:

$$q_s(0) = \frac{1}{2} \qquad q_s(R) = \frac{1}{2}(1 - 2F^2R) \tag{5.10}$$

and then the probability that it is still on after R new training episodes is:

$$p_s(R) = 1 - F^2R \tag{5.11}$$

In a similar manner to Eq 3.6, an upper bound can be placed on R, R_s, when the expected number of omission errors is 1:

$$M(1 - p_s(R_s)^M) = 1 \tag{5.12}$$

More typically, such a criterion for reliable retrieval is based on the total number of output errors. This total comprises both spurious and omission errors. Thus $H_L = 2$ would correspond to the constraint:

$$M(1 - p_s(R_s)^M) + (N - M)p_n(R_s)^M = 2 \tag{5.13}$$

However, when $p_n = p_c$ and $(N - M) \simeq N$, the second quantity is $\simeq 1$ from Eq 3.6, and so Eq 5.12 holds near enough.

Substituting in the expression for $p_s(R)$ allows an estimation of the survival time of an association:

$$R_s \simeq -\frac{N^2}{M^3} ln(\frac{M-1}{M}) \simeq \frac{N^2}{M^4} \quad \text{for } 0 \ll M \ll N \tag{5.14}$$

Span

If it were the case that every association had a survival time of exactly R_s, then the span, S, would be equal to R_s. This of course will never be the case, and in fact, the distribution of survival times follows an exponential decay, as in Fig 5–1 However, the many patterns with survival times less than R_s are accompanied by fewer patterns that have very long survival times ($> 2R_s$). Thus, average span S should correspond to expected survival time R_s.

If N is large and $M = log_2(N)$, the predicted span is of the order of:

$$O\left[\frac{N^2}{log(N)^4} \right] = O\left[\frac{R_c}{log(N)^2} \right] \tag{5.15}$$

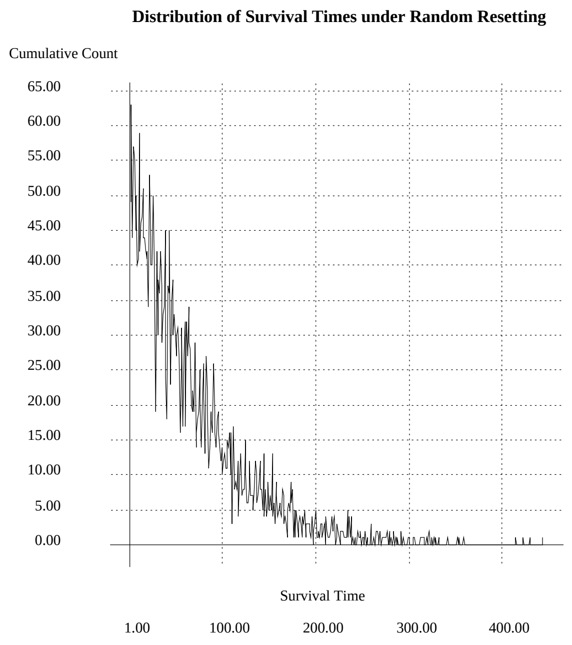


Figure 5–1: Example Distribution of Survival Times

Also note that, although it is stretching the sparse coding assumptions made above, a value of M as large as $\sqrt{N}$ would predict a span only of the order of 1. This reinforces the importance of sparse coding in most uses of the WN, including as a short-term memory.

Simulation with $F = 1/\sqrt{N}$ confirms this, giving spans of 2-3 pattern pairs.

5.1.2 General Case Analysis

The Method

Here, two new variables are introduced: z, the probability that a switch is triggered under appropriate conditions (formerly 1 in the above analysis) and τ, the threshold of the activation function (constant over output units for convenience - see [Willshaw & Dayan 90]).

Survival Time

The recurrence relation is now:

$$\begin{aligned}\hat{p}(t+1) &= \hat{p}(t)(1-r) + zF^2(1-\hat{p}(t)) \\ &= (1-r-zF^2)\hat{p}(t) + zF^2 \end{aligned} \tag{5.16}$$

Thus, after R training episodes:

$$\hat{p}(R) = (1-r-zF^2)^R\hat{p}(0) + \frac{zF^2}{r+zF^2}(1-(1-r-zF^2)^R) \tag{5.17}$$

The net stabilises at a loading $p = \hat{p}_\infty$, since starting with a Tabula Rasa ($\hat{p}(0) = 0$) and letting $R \to \infty$:

$$\hat{p}_\infty = \frac{zF^2}{r+zF^2} \tag{5.18}$$

Note that when $r = 0$, $\hat{p}_\infty = 1$, which results in the normal catastrophic failure. The interesting cases are only really when $0 < r < 1$ and $0 < z \leq 1$.

Again, consider the survival time of a switch triggered by a particular association learnt at time $t = 0$, such that $p_s(0) = 1-(1-p)(1-z)$:

$$\begin{aligned}p_s(R) &= (1-r-zF^2)^R(p+z-pz) + p(1-(1-r-zF^2)^R) \\ &= (1-r-zF^2)^R(1-p)z + p \end{aligned} \tag{5.19}$$

Since the threshold can be lowered below M, the reliable retrieval criterion from Eq 5.13 becomes:

$$M(1-P(p_s,\tau)) + (N-M)P(p_n,\tau) = 2 \tag{5.20}$$

where $P(p_x,\tau)$ is the probability of a weighted sum greater or equal to τ:

$$P(p_x,\tau) = \sum_{i=\tau}^{M} C_i^M p_x^i (1-p_x)^{M-i} \tag{5.21}$$

Span

When τ is (naively) retained at M, Eq 5.21 simplifies, and then expressing and maximising Eq 5.20 in terms of p, gives optimal $p \simeq (\frac{2}{N(M+1)})^{\frac{1}{M}}$ and hence optimal

r from $r = \frac{1-p}{p}F^2$. For a 512(9) net, this means an optimal $p = 0.44$. This in turn predicts a span of $S \simeq 57$, which is an improvement of $\simeq 130\%$ over the span with capacity-optimal $p = p_c$.

However, when $\tau < M$, a more sophisticated signal-to-noise analysis is needed. Such analysis soon gets very complicated however, and approximations are the only way to make the mathematics tractable. Discussion of the signal-to-noise ratio and possible approximations is given in Appendix B.

Direct Estimation

An alternative approach is to derive a general expression for the span of a net in terms of p_s and p_n. Since the span of a net is the number of patterns with $H_O \leq 1$ - *i.e.* with no errors, one omission error, or one spurious error - the probability of one of these situations arising, $Q(R)$, can be found, enabling a derivation for S:

$$\begin{aligned} S &= \sum_{R=0}^{R=W} Q(R) \\ Q(R) &= P_s^M (1-P_n)^{N-M} + \\ &\quad M P_s^{M-1}(1-P_s)(1-P_n)^{N-M} + \\ &\quad (N-M)P_s^M(1-P_n)^{N-M-1}P_n \qquad (5.22) \\ P_s &= P(p_s(R), \tau), \quad P_n = P(p_n, \tau) \end{aligned}$$

where $P(p, \tau)$ is defined in Eq 5.21, and W is the size of a "window" of pattern pairs which are likely to retrieved reliably. In principle, it is possible for $W \to \infty$, since even when R is large and $p_s \to p$, $Q(R)$ is still finite (but small) - a retrieval by "fluke" alone. In practice however, it is sufficient to just set a W large with respect to the expected survival time R_s.[1]

This expression for the span, together with equations for p_s and p_n, can only be studied by numerical analysis. Such number-crunching in fact gives good predictions for S, although they will always tend to be over-estimations because of its unit usage approximation.

[1]Experience shows that making W one order of magnitude larger than R_s is normally sufficient.

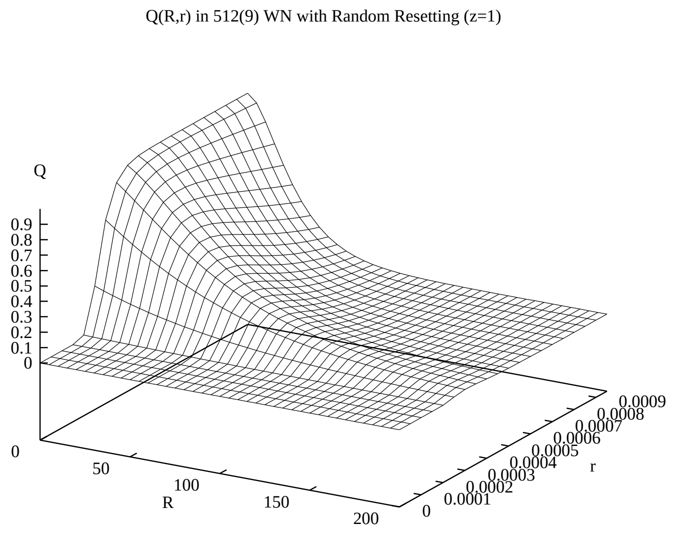


Figure 5–2: Probability of reliable retrieval of a pattern under Random Resetting from Numerical Analysis, as a function of r and R

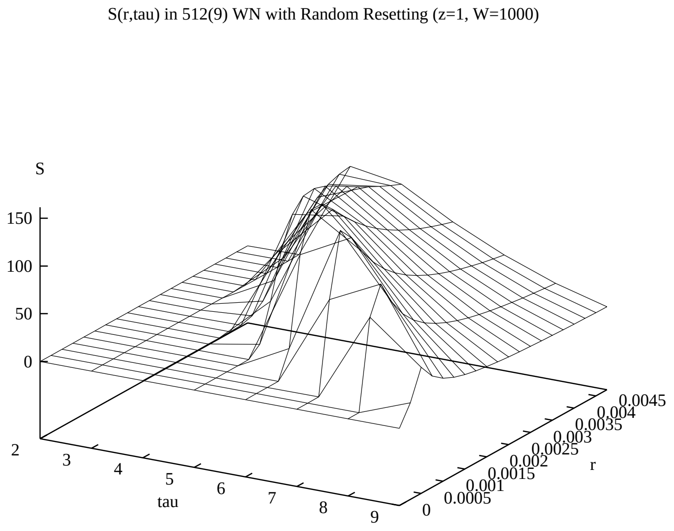


Figure 5–3: Span under Random Resetting from Numerical Analysis, as function of r and τ

Case	r$\times 10^{-3}$	(p)	τ	St	Sn	Sp
$p = p_c$	0.309	0.50	9	42.4	47.7	44.2 $\pm$5.5
τ=M	0.395	0.44	9	57.0	57.4	56.2 $\pm$4.8
Any τ	1.60	0.16	6	-	163	149 $\pm$7.4

Table 5–1: Maximum Spans for different Resetting Probabilities

Figs 5–2 and 5–3 show the functions $Q(R, r)$ and $S(\tau, r)$ from the results of numerical analysis. The theoretical maximum span obtainable for a 512(9) net is $\simeq 163$, which occurs when $r = 0.0016$ and $\tau = 6$.

5.1.3 Performance

Some simulation results for Random Resetting in a 512(9) *net are shown in Table 5–1. Fig 5–4 shows a span graph for the special case of Random Resetting. Here, average span is* 44.2 ± 5.5. *Fig 5–5 shows the stability imposed on* p *when Random Resetting is initiated after some initial training. Fig 5–6 shows the serial order curves from simulations with special case and optimal general case Random Resetting with* $W = 1,000$.

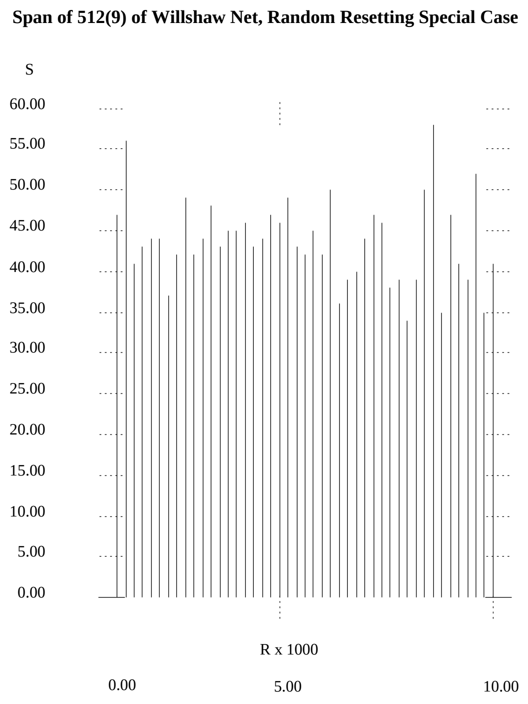


Figure 5–4: Span under Random Resetting Special Case

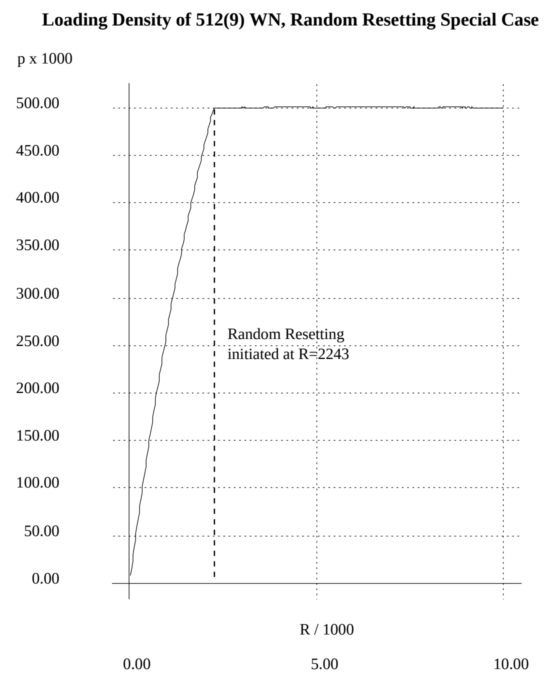


Figure 5–5: Loading Density under Random Resetting Special Case

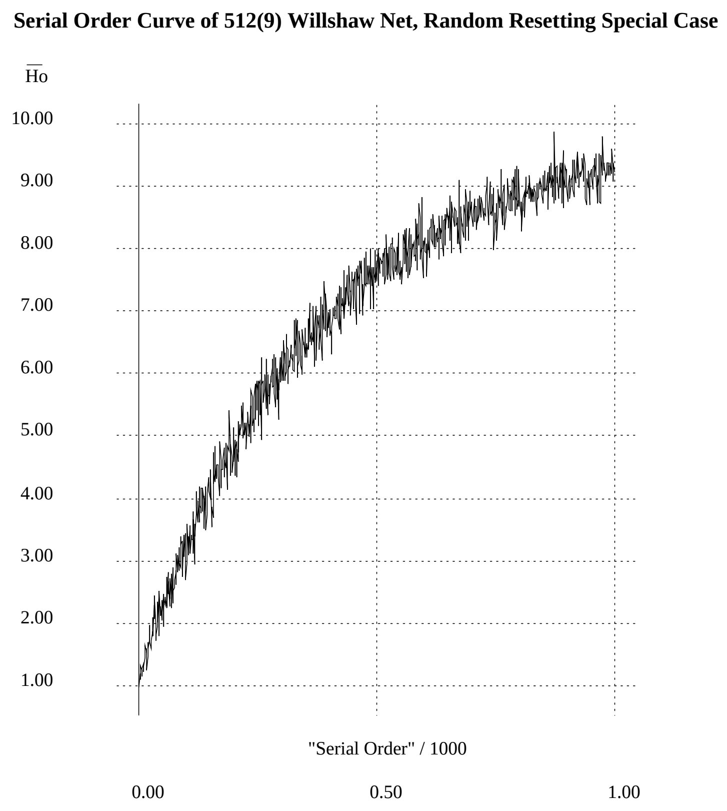
Span

In the table, the three rows correspond to the special case in section 5.1.1 when $p = p_c$, the greatest span when $\tau = M$ and the greatest span for any τ (all have $z = 1$). The columns show r, p (which is determined by r), τ and then three estimates of S: from theory, St, from numerical analysis, Sn, and from practice/simulation, Sp.

It can be seen that the three estimates of S are very close. The results for naive thresholding confirm R_s to be an accurate predictor of S - since theory matches practice to within one standard deviation.

The only significant discrepancy is for the optimal span under general Random Resetting, when numerical analysis would predict $S = 163$, whereas simulation results give $S = 148.9 \pm 7.4$ (theoretical prediction is unavailable here). This is most probably due to the unit usage assumption used in the numerical analysis.

Note the best possible span comes from a light loading, whence the expected number of spurious errors is $\simeq 0.46$, which is less than the expected number of omission errors (1.54 at the limit $R = R_s$). Given the low amount of information



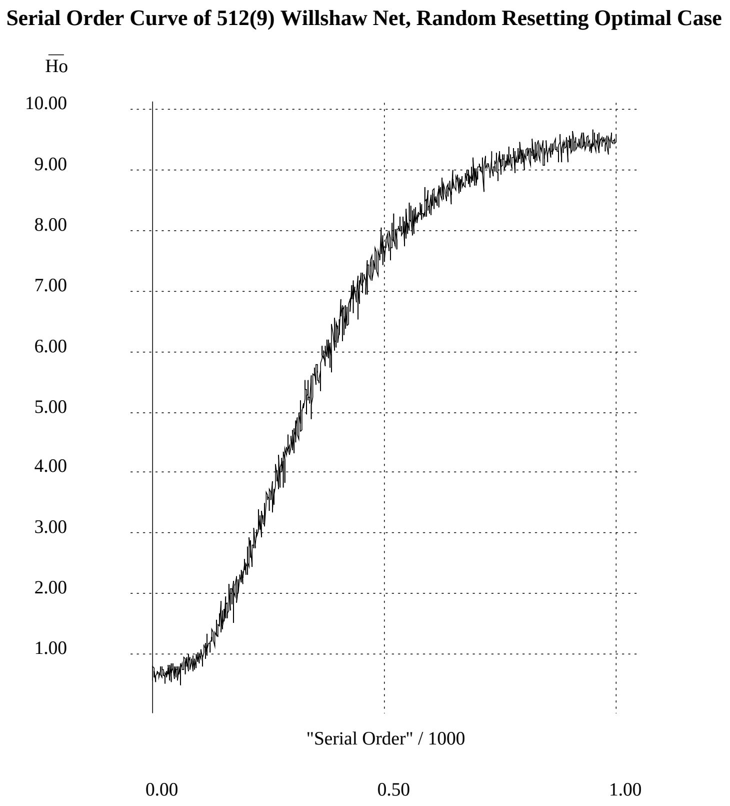


Figure 5–6: Serial Order Curves under Random Resetting

in sparsely coded patterns, this may make a common limit of H_L an unsatisfactory choice, but further consideration is not given here.

Serial Order Curve

From the serial order curves, it can be seen how older associations are progressively forgotten in an exponential manner - such a memory has gradual, “soft” degeneration with age. Of course, this entails a finite probability that even very recently trained patterns will fail to be retrieved reliably. The optimal case produces a better-defined serial order curve than the special case, as can be seen from comparing the two graphs.

It can be noted that R_s corresponds to the point on the serial order curve when $\overline{H_O} = 2$ (although the noise in the data makes accurate comparison impossible).

The asymptotic $\overline{H_O}$ value will be different for the two cases (though they appear similar). For the special case when $p = p_c$, the asymptote should be around 10, since there would be only one active output unit expected if an input pattern was presented to a completely random weight matrix. It may need a larger window to see this asymptote reached (p_s is probably still slightly larger than p_n), whereas the optimal case, being sharper, has definitely peaked by about 900 forgetting episodes.

5.2 Weight Ageing

The Method

This method turns off switches with a probability, $r(A)$, that is a function of the *age* of a switch, A. The age of a switch is the time (number of subsequent patterns trained) since that switch was *last* triggered. This includes “re-triggering” cases, when $w_{ij} = 1$ prior to training: then the switch age is “reset” by the new training episode.

To stabilise loading density, the new switches triggered in the learning of an association need to be accompanied by the resetting of an equal number of existing switches. The former quantity is again just:

$$M_I \times M_O \times (1-p) \tag{5.23}$$

Now one choice of $r(A)$ is a step-function parametrised by a "critical age" parameter, A_o, where $r(A) = 1 \; iff \; A \geq A_o$ (and 0 otherwise). Then the second quantity simply becomes the number of switches older than this critical age. Each such switch cannot have been re-triggered by any of the A_o learning episodes subsequent to its original triggering, so the requirement is that:

$$(1-p)M_I M_O = M_I M_O (1 - F_I F_O)^{A_o} \simeq M_I M_O \; \exp(-A_o F_I F_O) \tag{5.24}$$

Of course, this argument has followed a similar line to that producing Eq 3.5, and consequently the above requirement is satisfied when $A_o = -\frac{N_I N_O}{M_I M_O} ln(1-p)$.

In summary, an episode of forgetting in this "ideal" Weight Ageing algorithm involves a search through all triggered switches for switches that are older than A_o. Only these switches are reset.

A more general class of probability-age functions has $r(A)$ increasing smoothly with age. A sigmoidal function is such a function, which provides extra flexibility through a parameter D, determining the "sharpness" of the sigmoid (the Heaviside function can be viewed as an infinitely sharp sigmoid). Then:

$$r(A) = \frac{1}{1 + e^{-D(A-A_o)}} \tag{5.25}$$

The simulators allow experimentation with values for D as well as A_o.

Survival Time

With a discontinuous $r(A)$, the expected survival time will, of course, be equal to the number of pattern pairs with $H_O < H_L$ when $R = A_o$ under standard training. An optimal value for A_o can then be obtained from Fig 3–4, for the maximum ordinate value. In this case, A_o should be $\simeq 1900$. Survival times with such $r(A)$ functions give spans:

$$S = O\Big[\; (\tfrac{N}{log(N)})^2 \; \Big] = O[R_c] \tag{5.26}$$

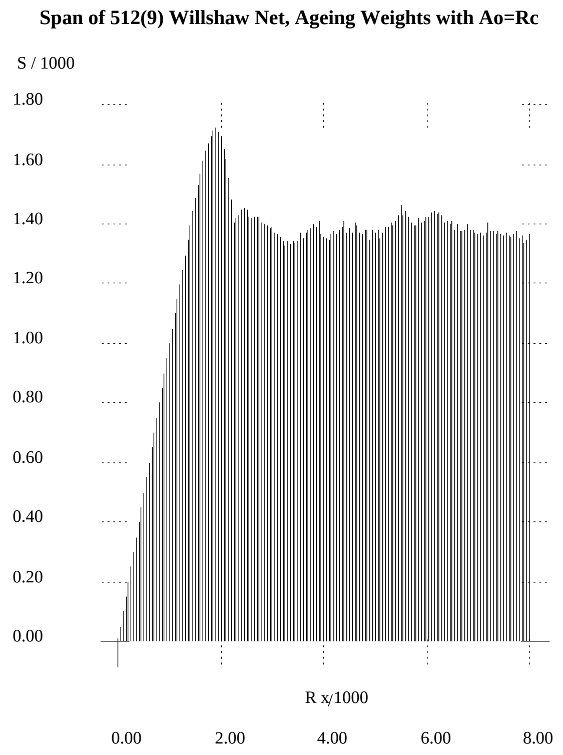


Figure 5–7: Span under Ageing Weights

5.2.1 Performance

Span

Fig 5–7 and 5–8 show the span and serial order curves for Weight Ageing in a 512(9) *net trained from a Tabula Rasa with a Heaviside r function and* $A_o = R_c$ *(*$p = p_c$*). The average span, after stabilisation, is* $(1.37 \pm 0.04) \times 10^3$.

The measured span is in agreement with the value of S from Fig 3–4 when $R = R_c$. If $A_o = 1900$, better spans of $(1.70 \pm 0.01) \times 10^3$ can be achieved. Interestingly, at this lower loading, a much more “stable” span results, as can be seen from comparing the standard deviations of the above two estimations.

For sigmoidal probability-age functions, as expected, the span decreases as the sharpness of the sigmoid decreases. However, there is little decrease until $D \ll 1$. When $D = 0.01$ for example, the span is still $(1.069 \pm 0.006) \times 10^3$.

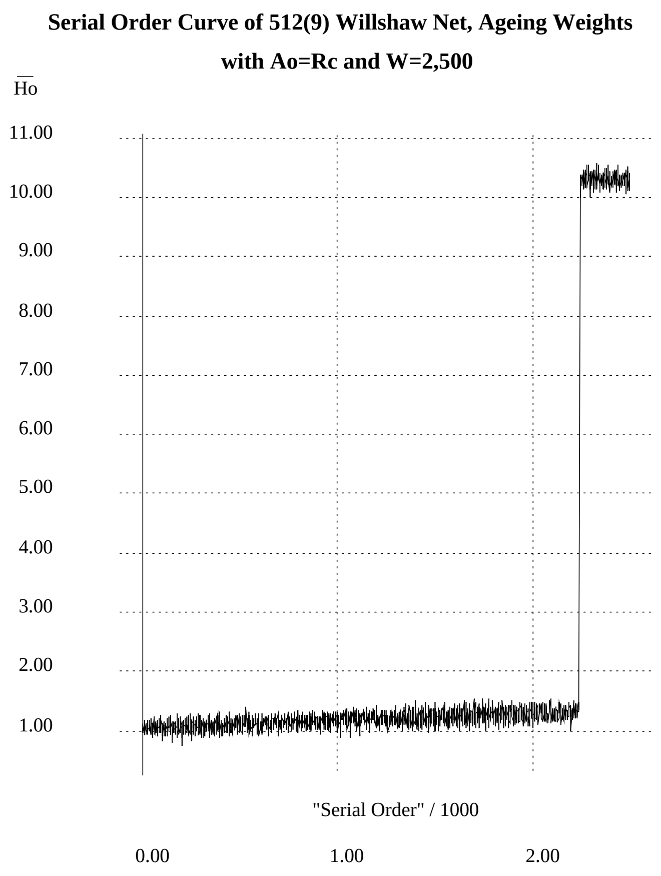


Figure 5–8: Serial Order Curve under Ageing Weights

Serial Order Curves

The discontinuous serial order curve shows how the more selective nature of the weight ageing method can produce almost “ideal” memory profiles.

	$v_j^{(O)} = 0$	$v_j^{(O)} = 1$
$v_i^{(I)} = 0$	w	y
$v_i^{(I)} = 1$	x	z

Table 5–2: Generalised Hebb Rule

5.3 Generalised Learning

Generalised Learning methods are not random, but "directed" in the sense that the switches reset are in some way dependent on the particular pattern vectors presented at time t. For example, whilst switches connecting conjoint activities might be triggered, the switches connecting active input units to *inactive* output units might be simultaneously reset.

5.3.1 Generalised Hebb Rule

Neurophysiological studies have shown that changes in synaptic efficacy would seem to be governed by quite complex processes. It has been suggested recently, *e.g.* [Stanton & Sejnowski 89], that increases in synaptic strength, or Long-Term Potentiation (LTP), may be balanced by decreases in efficacy through Long-Term Depression (LDP) - although the results are often in debate [Willshaw & Morris 89].

However, it is possible to explore an abstract space of possibilities - for example [Palm 88] draws up a generalised framework for weight update rules (for continuous weighted nets) as shown in Table 5–2. In Palm's formalism the variables $w - z$ correspond to real-valued weight changes.

[Dayan & Willshaw 91] determine optimum cases in this formalism, with respect to signal-to-noise ratios. The best learning rule turns out to be the Covariance rule, *e.g.* [Dayan & Sejnowski 93], studied in statistics (*c.f.* Eq 5.28), where w, z are increments to a weight and x, y are decrements.

In the sparse coding limit, other rules approach optimality: a Homosynaptic rule (when $F_O \to 0$, *c.f.* Eq 5.30), a Heterosynaptic rule (when $F_I \to 0$, *c.f.* Eq 5.32) and of course the standard rule used in the WN, which is non-optimal but best when both input and output patterns are sparsely coded (*c.f.* Eq 5.27).

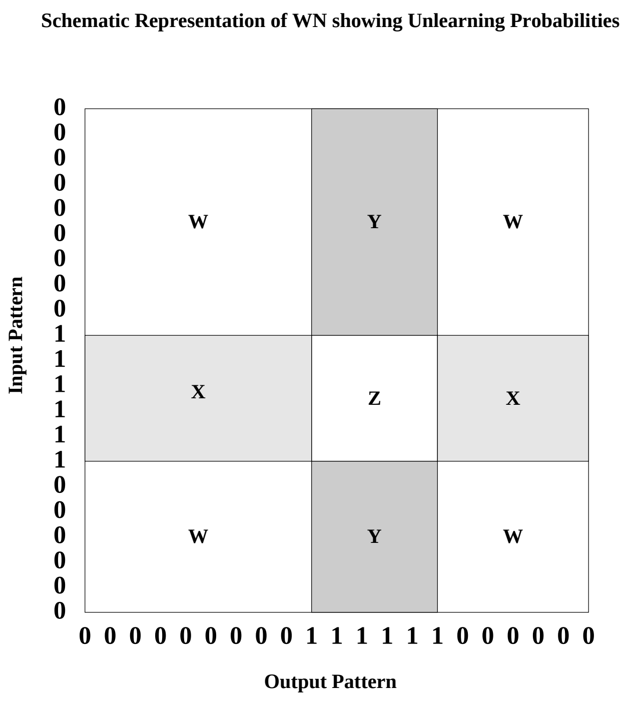


Figure 5–9: Schematic Illustration of Generalised Learning

Note that the standard HN learning rule has $x = y = -\eta$ and $w = z = +\eta$ (weights are only symmetrical when $x = y$). The HN learning rule only approaches optimality when the patterns are uniform signary vectors.

The Method

The possibility of resetting switches in a similar manner (or *unlearning*) suggests that modified learning rules could produce a stable WN loading below the asymptotic value of $p = 1$. This would be achieved by converting the weight increments or decrements for real-valued weights into probabilities for triggering or resetting binary-valued weights. Then the variables in Table 5–2 would correspond to:

$$\begin{aligned} w, z &= prob(w_{ij} \rightarrow 1) \\ x, y &= prob(w_{ij} \rightarrow 0) \end{aligned}$$

The relevant probabilities for switches are shown schematically in Fig 5–9. Below are listed some of the particular values for the probabilities $w - z$ investigated:

- Standard WN Learning rule:

$$w = x = y = 0, z = 1 \tag{5.27}$$

- Covariance Rule:

$$\begin{aligned} w = M_I M_O / N_I N_O, &\quad z = (N_I - M_I)(N_O - M_O)/N_I N_O, \\ x = M_I(N_O - M_O)/N_I N_O, &\quad y = M_O(N_I - M_O)/N_I N_O \end{aligned} \tag{5.28}$$

- Homosynaptic Unlearning:

$$\begin{array}{rlll} & w = y = 0, & x > 0, & z > 0 \\ Eg & w = y = 0, & x = M_O/(N_O - M_O), & z = 1 \quad (5.29) \\ Eg & w = y = 0, & x = M_O/N_O, & z = (N_O - M_O)/N_O \quad (5.30) \end{array}$$

- Heterosynaptic Unlearning:

$$\begin{array}{rlll} & w = x = 0, & y > 0, & z > 0 \\ Eg & w = x = 0, & y = M_I/(N_I - M_I), & z = 1 \quad (5.31) \\ Eg & w = x = 0, & y = M_I/N_I, & z = (N_I - M_I)/N_I \quad (5.32) \end{array}$$

- Keinosynaptic Unlearning:

$$\begin{array}{rlll} & x = y = 0, & \hat{w} > 0, & z > 0 \\ Eg & x = y = 0, & \hat{w} = M_I M_O/(N_I - M_I)(N_O - M_O), & z = 1 \quad (5.33) \end{array}$$

where $\hat{w} \neq w$, but here is the $prob(w_{ij} \to 0)$.

Survival Time

The survival time under Generalised Learning can be studied by similar methods to those in Section 5.1.2. Making the same simplifications for a square net as before, the following arise:

$$\begin{aligned} \hat{p}(t+1) &= \hat{p}(t)(1 - (x+y)F(1-F)) + (1 - \hat{p}(t))(w(1-F)^2 + zF^2) \\ &= (w(1-F)^2 + zF^2) + (1 - (w(1-F)^2 + (x+y)F(1-F) + zF^2))\hat{p}(t) \end{aligned}$$

$$p = \frac{w(1-F)^2 + zF^2}{w(1-F)^2 + (x+y)F(1-F) + zF^2} \tag{5.34}$$

$$\hat{p}(R) = p + (1 - (w(1-F)^2 + (x+y)F(1-F) + zF^2))^R(\hat{p}(0) - p) \tag{5.35}$$

Letting $k = 1 - (w(1-F)^2 + (x+y)F(1-F) + zF^2)$, a recurrence relation for p_s can be derived similar to the general Random Resetting case:

$$p_s(0) = 1 - (1-p)(1-z) \tag{5.36}$$

$$p_s(R) = k^R(1-p)z + p \tag{5.37}$$

(The signal-to-noise ratio, ρ, will also be determined in a similar manner to Eq B.2 in Appendix B.) However, there are interesting differences to Random Resetting, particularly when p_n can no longer be approximated by p. These are highlighted by considering some of the examples in Eqs 5.29 to 5.28.

5.3.2 Analysis of Special Cases

The values of x and y in Eqs 5.29, 5.31, or 5.33 are derived by requiring a stable net loading of $p = p_c$ and $z = 1$, which suggests the optimal threshold is $\tau = M$. Then, since $w(1-F)^2$, $(x+y)F(1-F)$, zF^2 are all $\ll 1$, the recurrence relation becomes:

$$p_s(R) \simeq 1 - \frac{M^2}{N^2}R \tag{5.38}$$

This equation is of course identical to 5.11. Because patterns are randomly generated, learning new pattern pairs effectively results in Random Resetting of switches important to previous associations.

The value of $p_n(R)$ however, depends on the particular type of unlearning.

Homosynaptic

Consider Eq 5.29: now, immediately after learning a given association:

$$p_n(0) = p(1-x) \tag{5.39}$$

After training R further associations, $p_n(R) \rightarrow p$, being the stable point of the recurrence relation from 5.35 again (this time for p_n):

$$p_n(R) = p - (1 - (xF(1-F) + zF^2))^R px \tag{5.40}$$

In this example, where $p = 0.5$ and $x = \frac{F}{1-F} \ll 1$:

$$p_n(R) \simeq \frac{1}{2}(1 - 2\frac{M^3}{N^3}R) \tag{5.41}$$

Substituting $p_n(R)$ and $p_s(R)$ from 5.38, the following approximation is derived for R_s (from 5.13):

$$R_s \simeq \frac{1 + \frac{M^2}{N}}{\frac{M^4}{N^2} + \frac{2M^4}{N^3}} = O\left[\frac{N^2}{log(N)^4} \right] \tag{5.42}$$

Though this is of the same order as Eq 5.14, it suggests a survival time slightly greater than with Random Resetting (and results confirm this - see later).

Another advantage of such a method would be apparent if the sequences of pattern pairs trained were not random, but correlated in some way. For example, if input patterns were all drawn from a similar template (*i.e.* were separated by small hamming distances - sharing 1's in several places), the span of a net under Random Resetting would fall dramatically, as the number of spurious errors would increase. However, under homosynaptic unlearning, recently trained patterns are more thoroughly "impressed" on the memory, reducing the chance of spurious errors.

This is supported by simulation results, where correlated patterns can be produced by setting components to 1 *with differing probabilities. In a* 512 *net with* $E[M] = 9$*, but with* 9 *of the components* 30 *times more likely to be set than the other* 503*, the average span under Random Resetting is* 1.5 ± 1.3*, whereas under Homosynaptic Unlearning, the average is* 13 ± 3.3.

Heterosynaptic

Since this method does not affect weights responsible for spurious errors like homosynaptic unlearning, the value of $p_n \simeq p$. However, the method should damp the fluctuations in unit usage, since any increase in the usage of a particular unit, $zM(1-p)$, is now countered by a decrease of $(N-M)y$.

Indeed, this is observed in practice, when, for example, the variance of unit usage in a 512(9) *net under heterosynaptic unlearning is 4 times smaller than under homosynaptic.*

Note that the distinction between homo- and heterosynaptic unlearning would be irrelevant if retrieval of patterns in a WN could procede in either direction. In other words, if "output" patterns could cue retrieval of "input" patterns, as well as vice versa, then it would be advantageous for a palimpsest scheme to employ both homo- and heterosynaptic methods.

Keinosynaptic

Keinosynaptic unlearning involves turning off switches if neither of the respective input or output units are active (*i.e.* the probability $\hat{w}$ is not the same as employed in the other rules). Since this method would seem to have neither of the advantages of the above two, it might be expected to perform no better than Random Resetting in the sparse coding limit. Indeed, as this ideal is compromised, keinosynaptic unlearning would probably perform worse than Random Resetting, since it actually encourages variability in unit usage. The only possible advantage of this scheme would be increased robustness to spurious active units in noisy cues, but this would have negligible effect with sparse coding, so has not been explored here.

Other Rules

Eqs 5.30, 5.32 and 5.28 are probabilistic equivalents of optimal cases of the Generalised Hebb Rule studied in [Dayan & Willshaw 91]. All three have $z < 1$, but still give a stable net loading of $p = 0.5$. As a consequence, all give the largest spans when $\tau = M$.

However, although the original rules can be shown to give optimal signal-to-noise ratios, which are essential for good span, they are not optimal for palimpsest-like behaviour with respect to S. This is likely to be for two main reasons.

Firstly, they are only really applicable when the weights are continuous, and conversion to probabilities in a binary-weighted net is not likely to retain their optimality. For example, there is no possibility for binary weights to take negative values, and reduce the weighted sum.

Secondly, the threshold here is identical for all output units, whereas the [Dayan & Willshaw 91] analysis assumes each unit can independently adjust its own threshold from knowledge of its quenched weights. A common threshold will not produce spans as good as with adaptable thresholds.

5.3.3 General Case Analysis

As with the general case of Random Resetting, better spans can be achieved when τ is allowed to drop below M, and the net is loaded more lightly. The threshold is then variable in the sense that a good choice will depend on the variables w, x, y, z.

When $\tau < M$, a mathematical expression for R_s is again hard to ascertain. However, the parameter space of w, x, y, z and τ can be again explored by numerical analysis from Eqs 5.22, 5.37 and 5.40, as in the "Direct Estimation" of Section 5.1.2. The results of such analysis are compared with simulation results in Table 5–3.

5.3.4 Performance

Span results from a 512(9) *net under various Generalised Learning schemes are shown in Table 5–3.*

Span

The first six rows in the table show particular values for w, x, y, z from Eqs 5.28 to 5.33. These values uniquely determine p from Eq 5.34. The column under Sn represents the maximum theoretical span obtained in numerical analysis from varying τ, whose maximising value is shown in the previous column. Sp stands for the span found in practice, from simulation.

The last two rows show maximal theoretical spans after exploring a space of values for w, x, y, z, τ by numerical analysis. For naive thresholding where $\tau = M$, a maximum $S \simeq 60$ can be obtained with $w = y = 0$, $z = 1$ and $x = 0.0216$. For a

Equation	$w/10^3$	$x/10^2$	$y/10^2$	z	p	τ	Sn	Sp
5.29	0.0	1.79	0.0	1.0	0.50	9	52	53.1 $\pm$5.2
5.30	0.0	1.76	0.0	0.982	0.50	9	33	31.8 $\pm$5.2
5.31	0.0	0.0	1.79	1.0	0.50	9	48	50.0 $\pm$5.6
5.32	0.0	0.0	1.76	0.982	0.50	9	30	31.9 $\pm$5.2
5.33	($\hat{w}$) 0.3	0.0	0.0	1.0	0.50	9	-	42.3 $\pm$5.2
5.28	0.3	1.73	1.73	0.965	0.50	9	11	11.4 $\pm$3.0
τ=M	0.0	2.16	0.0	1.0	0.45	9	60	59.0 $\pm$5.3
any τ	0.0	8.75	0.0	1.0	0.17	6	178	168 $\pm$7.3

Table 5–3: Maximum Spans for different Generalised Learning Methods

smaller threshold of 6, the much better span of $\simeq$ 178 is obtained, with $x = 0.0875$ (± 0.0025) instead.

The following are worth noting:

- The prediction from Eq 5.42 for S under homosynaptic unlearning, 46.3, is actually considerably smaller than that observed in practice (53.1 ± 5.2). The difference is due to the approximations made in the prediction.

- Both homosynaptic and heterosynaptic unlearning with $p = p_c$, give significantly larger spans than Random Resetting (t-tests: [$N = 100$, one-tailed; Homosynaptic: $t = 12.4$, $prob < 0.005$, Heterosynaptic: $t = 7.4$, $prob < 0.005$]). The difference for Homosynaptic unlearning is $\simeq 8.9$. Thus both reducing noise in spurious output lines and reducing the variability in unit usage confer significant increases in span.

- Results with keinosynaptic unlearning, as expected, give a span significantly down on the Random Resetting case (t-test: [$N = 100$, $t = 3.1$, $prob < 0.005$, one-tailed]), with a difference of 1.9.

- The effects of $w > 0$ or $z < 1$ (as with Random Resetting) are insignificant versus the effects of homo- and heterosynaptic unlearning.

- Although a finite y is advantageous, its value is out-weighed by having x large enough for an optimal τ and $p \simeq 0.17$.

- Optimal loading densities and thresholds under Random Resetting and Generalised Learning are very similar. This is to be expected, given the only effective difference is in the noise term p_n. However, this may not be the case if patterns were correlated or pattern coding was less sparse.

Palimpsest Scheme	Optimal Parameters	Span (approx)
Random Resetting	$r = 0.0016$, $z = 1$, $\tau = 6$	149
Weight Ageing	$r(A) = 1 \leftrightarrow A > 1900$	1,700
Generalised Learning	$x = 0.088$, $z = 1$, $w = y = 0$, $\tau = 6$	168

Table 5–4: Comparison of Palimpsest Schemes in Willshaw Net

5.4 Summary

All the palimpsest schemes discussed above, given certain choices of parameters, can allow a net to function as a short-term memory with a stable span. Given an optimal choice of these parameters (found by numerical analsysis when $N = 512$ and $M = log_2(N) = 9$), the size of this span is shown in Table 5–4.

Chapter 6

Forgetting in the Hopfield Net

This chapter examines four main proposals for palimpsest schemes in the Hopfield Net: *bounded weights*, *attentuated weights*, *random unlearning* and *enforced storage*. Approximate predictions of the noise-free spans under each method are derived via a statistical analysis of weight variablility and behaviour of local fields.

6.1 Catastrophic Failure

6.1.1 Recap

As mentioned in Chapter 3, a HN loaded beyond α_c fails completely as an associative memory. In energy terms, this is because the basins of attraction created by learnt patterns interfere and distort, resulting in instability and the appearence of spurious attractors. The point of failure can be approximated by considering the behaviour of local fields; a perspective offered in Section 4.3.2.

This failure may also be studied from a slightly different perspective, that of the weights themselves. Put simply, since the growth of weights is unbounded, the contribution of any learnt pattern to the nature of **W** diminishes, and it no longer becomes possible for learning to create stable states. This alternative perspective is outlined first, before both are employed in examination of palimpsest schemes.

6.1.2 Analysis of Weight Changes

Consider the continuous training of a HN under a palimpsest scheme. Learning of a new pattern $\mathbf{v}^{(p)}$, after $R = p - 1$ have already been trained, is achieved through Eq 4.1:

$$w_{ij}(p) = w_{ij}(R) + \Delta w_{ij}(p) \tag{6.1}$$

where:

$$\Delta w_{ij}(p) = \eta v_i^{(p)} v_j^{(p)} \tag{6.2}$$

From Eq 4.7, the condition that this pattern be an initially stable state of the net is:

$$\forall i \left[\textstyle\sum_{j=1}^{N} w_{ij}(R) v_j^{(p)} v_i^{(p)} + \sum_{j=1}^{N} \Delta w_{ij}(p) v_j^{(p)} v_i^{(p)} \right] > 0 \tag{6.3}$$

For randomly chosen patterns, the first quantity is a sum of N independent, random variables between $-w_{ij}(R)$ and $+w_{ij}(R)$, with a mean of 0. If N is large enough, the variance of each variable will be approximated by $K(R)$, where:

$$K(R) = \overline{w_{ij}^2} - \overline{w_{ij}}^2 \tag{6.4}$$

where the bar means averaging over all R patterns. Then the standard deviation of the first sum, obtained from the sum of the variances, is $\sqrt{NK(R)}$.

Treating the Δw_{ij}'s also as independent random variables, the second sum is simply equal to $(N-1)\eta$ (since $w_{ii} = 0$). Hence the pth pattern is likely to be a stable attractor if η is greater than some critical value:

$$\eta > \varepsilon \sqrt{\frac{K(R)}{N}} \tag{6.5}$$

where ε is a numerical factor.

Under standard learning, the variance of the w_{ij}'s is the sum of the variances of each weight change, which is η. Hence, starting from a Tabula Rasa, where $K(0) = 0$:

$$K(R) = R\eta^2 \tag{6.6}$$

The condition that 6.5 be fulfilled is then:

$$\frac{R}{N} < \frac{1}{\varepsilon^2} \tag{6.7}$$

Since this criterion is relevant to any of the R learnt patterns, as soon as it is exceeded by training more patterns, all such patterns are simultaneously forgotten - hence the catastrophic failure found in practice. From section 4.3, the estimate of $\alpha_c = 0.14$ gives $\varepsilon \simeq 2.6$.

Therefore, to control this overloading effect, the acquisition the latest pattern (signal) must be controlled relative to the background noise K.

One way is to make η an exponentially increasing function of R. Then the signal strength, $\eta(R)$, grows with R (as does the noise), and Eq 6.6 becomes:

$$K(R) = K(0) + \sum_{p=1}^{R} \eta(p)^2 \tag{6.8}$$

By considering a continuous behaviour of these quantities, [Nadal *et al* 86] and [Nadal 86], derive a *marginalist* function for $\eta(R)$, such that it is always tuned exactly to stabilise the last pattern trained:

$$\eta(R)^2 = \eta(0)^2 \exp \frac{\varepsilon^2 R}{N} \tag{6.9}$$

However, this unbounded growth of η makes the palimpsest implausible as a short-term memory device operating indefinitely (as $R \to \infty$). A better way to achieve this is by either placing an upper limit on K or attempting to keep it constant.

6.2 Bounded Weights

The Method

[Hopfield 82], [Sompolinsky 86] suggest a non-linear learning function, where the range of values that can be taken by w_{ij} is limited in $B \leq w_{ij} \leq B$. This in turn places an effective upper limit on $K = B^2$. The learning rule is now:

$$w_{ij}(p) = g(w_{ij}(R) + \eta v_i^{(p)} v_j^{(p)}) \tag{6.10}$$

where:

$$g(x) = \begin{cases} -A & \text{if } x \leq -B \\ x & \text{if } -B < x < B \\ A & \text{if } x \geq B \end{cases}$$

(although any other non-linear function $g(x)$ with saturation would suffice).

The value of B can be chosen such that the distribution of w_{ij}'s should still be seriously affected when $\frac{R}{N} = \alpha_c$. This means B cannot be much bigger than B_c where:

$$B < B_c = \sqrt{N\alpha_c\eta^2} = \frac{\eta}{\varepsilon}\sqrt{N} \tag{6.11}$$

For much smaller B, only the most recent pattern is stored. For much larger B, the bounds have no effect and catastrophic failure still occurs.

Spans

When η is greater than some critical value, the net obtains a stationary span. For such a stationary regime $\eta > \frac{2.7}{N}$ $(B = \frac{1}{\sqrt{N}})$. This critical value has been obtained by Markov chain representation of the iterative training [vanHemmen *et al* 88] or by viewing the weights as performing random walks between reflecting barriers: in terms of approximate signal-to-noise ratios [Gordon 87], or as an exact solution in sparsely-connected nets [Derrida & Nadal 87].

The optimal value of η is $\simeq \frac{3}{N}$ [Gordon 87]. This is in excellent agreement with the optimum found from numerical analysis [Nadal *et al* 86]. Spans where the retrieval quality $m_o > 0.97$ are then obtainable: $S \simeq 0.04N$, *i.e.* still of $O[N]$, but now a third as much as $\alpha_c N$.[1]

[Parisi 86] effectively plots a serial order curve for this type of bounded learning in which the probability of reliably retrieving a pattern shows a smooth decay as patterns get older. For N in the range $200-400$, 70% of the total storage capacity has this probability at 90% or higher. More detailed investigations are needed to see if the sigmoidal function approaches a step-function in the large N limit.

Note that if the bounds are "absorbing" rather than reflecting (*i.e.* if $w_{ij}(p) = \pm B$, then $w_{ij}(p+i) = \pm B$ $\forall i$ subsequent patterns), then the net reaches an asymptotic state when no new patterns can be learnt, but the oldest patterns remain stable attractors and there is no failure of these memories. This has been compared to a fixed capacity Long-term Memory [Peretto 86].

[1]Finally, when η is increased beyond $N^{-\frac{1}{2}}$, the span is exactly 1.

[Amit & Fusi 92] suggest a slightly different bounding scheme where continuous weights undergo exponential decay over time, although periodically, they are "refreshed" to a finite set of *clipped* values. The frequency of refreshing is varied relative to the frequency of pattern training. When the frequencies are similar, this scheme provides a palimpsest with a span of $O[ln(N)]$. As the rate of presentation is increased, the span increases rapidly to orders $O[\frac{N}{ln(N)}]$. Alternatively the refresh mechanism can operate stochastically, which allows the pattern presentation to be slowed down significantly, but the span still cannot surpass $\sqrt{N}$.

6.2.1 Performance

Results from a $512(256)$ *HN with* $\eta = 0.00586$ *and weights bounded at* $B = 0.0442$ *give a span of* $S = 24.4 \pm 3.2$. *Fig 6–1 shows the serial order curve with the overlap metric. The mean weight value is* 0 *with variance as small as* 0.17η, *which is about* 75% *of the upper limit* B^2.

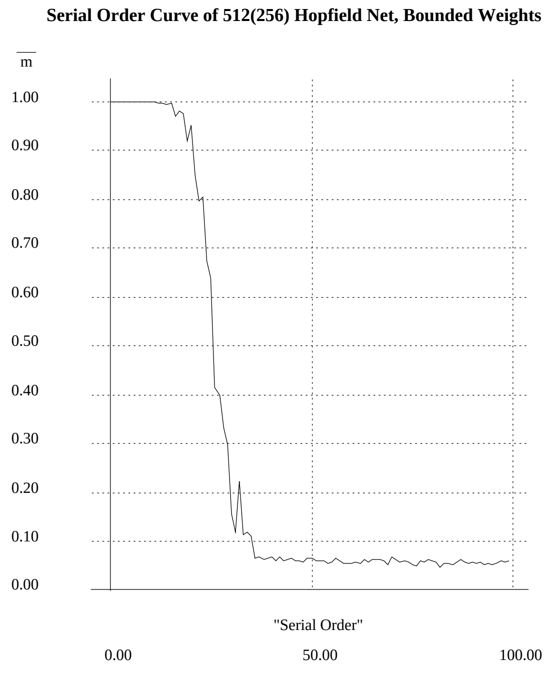


Figure 6–1: Serial Order Curve under Bounded Weights

The experimental S value is close to the predicted $S \simeq 20.5$ (given the variability of the simulated estimate). The sigmoidal serial order curve is quite sharp suggesting a sudden transition from remembering to forgetting - from $\overline{H_O} = 0$, to $\overline{H_O} \simeq 250$.

6.3 Attentuated Weights

The Method

[Nadal *et al* 86] also invent a new learning rule that keeps the value of K fixed at $K = \frac{1}{N}$. This is:

$$w_{ij}(p) = \lambda(w_{ij}(R) + \eta v_i^{(p)} v_j^{(p)}) \tag{6.12}$$

In a similar manner to Eq 6.8, this requires:

$$\lambda = (1 + N\eta^2)^{-\frac{1}{2}} \simeq exp\frac{-N\eta^2}{2} \tag{6.13}$$

in the large N limit. Thus each time a new pattern is learnt, the existing weights are attenuated by a fixed amount - they “decay” back to their mean value of 0. In other words, the creation of each new energy minimum is supplemented by the flattening of old ones.

Spans

As in section 4.3.2, the condition for a pattern $\mathbf{v}^{(p)}$ to be reliably retrieved after $R + 1$ have been trained is when:

$$\sigma \leq x_0 \overline{h}^2 \tag{6.14}$$

and in this scheme:

$$\overline{h_i(\mathbf{v}^{(p)})} = N\eta\lambda^{R-p} v_i^{(p)} \qquad \sigma = N\eta^2 \sum_{r=1}^{R} \lambda^{2(R-r)} \tag{6.15}$$

Therefore, if the last pattern must be well retrieved ($p = R+1$), and using Eq 6.13:

$$\frac{1}{N(1 - \exp(-N\eta^2))} \simeq \frac{1}{N^2\eta^2} \leq x_0 \tag{6.16}$$

This means that $\eta \geq \frac{2.56}{N}$. Alternatively, adopting the stronger condition that the last S patterns are well retrieved:

$$N\eta^2 S = ln(N^2\eta^2 x_0) \tag{6.17}$$

Maximising S with respect to η^2 gives the optimal η and span:

$$\eta = \sqrt{\frac{e}{x_0 N^2}} \simeq \frac{4.2}{N} \qquad S \simeq 0.05N \tag{6.18}$$

Thus the optimal span is slightly better than with Bounded Weights, although it is still approximately a factor of 3 smaller than $\alpha_c N$.

[Mezard *et al* 86] generalise this learning rule to a whole family of models. They solve the thermodynamics of this family in a similar way to the Hopfield model. Their calculation of the threshold for a stationary capacity is $N\eta > 2.46$, which is close to the above prediction, as is their optimal $N\eta = 4.11$.

A similar family of models has been proposed by [Shinomoto 87] for use in Boltzmann Machines, where forgetting involves similar attenuation, but the learning rule involves averaging over states organised by the stochastic rule implemented in the connection itself. However, in the zero temperature limit, the rule is formally equivalent to that of [Nadal *et al* 86].

6.3.1 Performance

Results from a 512(256) *HN with* $\eta = 0.00803$ *and Attenuated Weights with* $\lambda = 0.984$*, give a span of* $S = 29.9 \pm 1.9$*. The mean weight value is* 0 *with variance as small as* 0.25η*.*

This is actually over two standard deviations larger than the approximate theoretical prediction: $S \simeq 25.6$. This may be due to finite N effects again, or the inaccuracy of the above analysis (this is why $S = 0.05N$ is only given to one significant figure).

6.4 Random Unlearning

The Method

Another suggestion for avoiding catastrophic failure is to have some unlearning of attractors in the net. For example, this might involve starting the net in some random state and allowing it to relax to the nearest attractor. If that attractor corresponds to a state $\mathbf{s}'$, then unlearning of that state is simply the opposite process to the normal HN adjustment:

$$\Delta w_{ij} = -\epsilon s_i' s_j' \tag{6.19}$$

where ϵ is some constant.

This method was originally suggested by [Hopfield *et al* 83] as a method of improving the accessibility of trained patterns (but not as a palimpsest scheme). When $\epsilon < \eta$ and a considerable number of unlearning trials, u, are performed, unlearning improves memory function both by equalisation of the size of trained attractors and by suppression of spurious attractors. (This method has been compared to hypothesised physiological processes that may occur during REM sleep in animals, whereby random activity leads to weak and insignificant engrams being erased and stronger engrams actually being made more resilient.[2])

The idea has been further studied by [Kleinfeld & Pendergraft 87]. With a small net of 30 units they found that, with $\epsilon = \frac{1}{N}$ and $R = 11$ patterns trained, a massive improvement of $\simeq 25\%$ of patterns stored with no unlearning, to $\simeq 95\%$ stored when 120 unlearning trials were performed. This resulted in an almost two-fold increase in capacity of the net in information-theoretic terms. Providing $\epsilon < \frac{1}{R}$, the improvement is relatively independent of ϵ given a constant total amount of unlearning $u\epsilon$.

[2][Geszti & Pazmandi 87] give another interpretation of dream sleep, where learning is within bounds, *á la* Parisi, but the attractors that random initial states settle into are not unlearnt, but rather *relearnt*. This strengthening of strong memories within a bounded system automatically entails the weakening of more superficial (spurious) memories.

Spans

For a palimpsest scheme, the desire is to actually remove old pattern attractors as well as spurious attractors. This necessarily involves a more radical alteration of the energy landscape. Given alternating learning and unlearning episodes, this would be trivial with $S = 1$ and $u = 1$; then $\epsilon = \eta$. However, it should be very hard to obtain a stable span when $S > 1$.

This is because a pattern attractor needs a large amount of unlearning, $u\epsilon$, to smooth it out (else the normal interference will eventually occur). This means it will either have to be selected many times as $\mathbf{s}'$, or ϵ will have to be large. However, to be selected many times, S needs to be small, whilst if ϵ is large and $\mathbf{s}'$ corresponds to a spurious attractor instead, the unlearning will affect several other desired attractors too. This catch-22 situation seems irresolvable, as simulation results would support.

Alternatively, unlearning can take place after every e learning trials, where $e > 1$. If K is to remain constant, and it is assumed that $\mathbf{s}'$ is always a pattern attractor (*i.e.* $S \ll \alpha_c N$ - so unlearning will always decrease noise), then the requirement would be for $e\eta = u\epsilon$.

However, care must be taken when pursuing this idea. If $S \simeq e$, then much random unlearning simply involves "wiping the slate" clean every few training episodes. In the extreme case, this could produce an "span" of $S = \alpha_c N$ if $e = \alpha_c N$ and $u\epsilon$ was sufficient to mean that the subsequent training started from a Tabula Rasa again. The drawback is that every pattern is not treated equally; their survival times would be dependent on the time they were trained. Some patterns would have a survival time only of one training episode and the serial order curve would be totally flat.

Thus the choice of e relative to expected S effectively boils down to a choice on the form of serial order curve. Here, consideration shall only be given to $e < S$, to allow reasonable serial order curves.

6.4.1 Performance

Values for e, u and ϵ have been explored by simulation. Table 6–1 shows the

R_p	e	u	ϵ	Sp
1	1	1	η	1
2	1	1	η	2
3	1	1	η	0
2-8	1	10	0.1η	1
2-8	1	100	0.01η	1
2	2	2	η	1.5
3	3	3	η	0
2	2	20	0.1η	1.5
3	3	30	0.1η	2
4	4	40	0.1η	2.5

Table 6–1: Random Unlearning Results

results of S from a 512(256) net under different amounts of unlearning (which always precedes learning).

The first column in Table 6–1 under R_p gives the number of pre-learnt patterns before unlearning first begins. Sp is the span from simulations, after the (significant) initial transients have died down.

The main conclusions are then:

- With alternating learning and unlearning, the maximum $S \simeq 2$. When as few as 3 attractors exist in the net, there must be the creation of at least one spurious attractor.[3] This is because, when $\mathbf{s}'$ corresponds to a spurious attractor, the unlearning with $\epsilon = 1$ is likely to be too great and in effect causes the partial training of a new pattern, rather than the desired flattening of a pattern attractor. From this point onwards, the introduction of further spurious attractors is likely to escalate, and the palimpsest fails from normal interference - not even the latest pattern can be made stable.

- Reducing ϵ and increasing u means unlearning is likely to diminish several attractors in the net. Now several patterns may become unstable and forgotten. In the long-term, the best hope is a span of 1 from the normal learning

[3]The first spurious attractors to appear are co-called *mixture states*, which are linear combinations of an odd number of learnt patterns - see [Hertz *et al* 91]

of the last pattern (since the above "excess" learning is less likely to occur when $\epsilon < \eta$).

- The last few rows of the table show how it does not appear possible to obtain $S > e$ when $e > 1$. This is mainly due to $\mathbf{s}'$ occasionally being a spurious attractor (*i.e.* failure of the assumption in the previous Section). Though some patterns (those learnt immediately after the unlearning episode) have a survival time of e, *i.e.* greater than the average S, the differential treatment of patterns is not really satisfactory, as explained above.

Consequently, Random Unlearning <u>can</u> allow continual training, but is <u>not</u> an effective palimpsest scheme at all, since the best stable span obtainable is that of 2 patterns, and this would seem independent of N (since there is always a finite chance of a spurious attractor when $S > 2$ that can potentially destabilise the palimpsest).

6.5 Enforced Storage

The Method

[Morris & Wong 88] propose a learning rule that goes beyond the usual Hebbian rules. Their rule is based on the error-correcting properties of "delta-rules" like the Widrow-Hoff rule:

$$w_{ij} \to w_{ij} + \eta \Delta_i v_j \tag{6.20}$$

where Δ_i is a measure of error. The usual linear measure is simply the difference between the new pattern component $v_i^{(p)}$ and unit i's local field after presenting the pattern on the net just before it is learnt:

$$\Delta_i(p) = v_i^{(p)} - \sum_{k=1}^{N} w_{ik}(R) v_k^{(p)} \tag{6.21}$$

The most important result about these rules in the heteroassociative case is that $\mathbf{W}$ converges to the matrix that results in the minimal least square error for the estimation of the output pattern from the input pattern [Stone 86]. In the auto-associative case, this matrix is just $\mathbf{I}$. In return for the added complexity of the delta-rules, they no longer impose any requirement of orthogonality or even linear independence of pattern vectors in achieving their goals.

The drive behind the [Morris & Wong 88] rule is to "enforce" each new pattern on the net with sufficient intensity that it is always stored (*i.e.* the requirement given by Eq 6.3). They show this is the case if the learning rule is:

$$\begin{aligned}
w_{ij}(1) &= \frac{\eta}{N} v_i^{(1)} v_j^{(1)} && (6.22)\\
w_{ij}(p) &= w_{ij}(R) + \Delta w_{ij}(p) && \{i \neq j,\ p > 1\}\\
\Delta w_{ij}(p) &= \frac{1}{N}\left(\eta v_i^{(p)} - \textstyle\sum_{k=1}^{N} w_{ik}(R) v_k^{(p)} \right) v_j^{(p)} && (6.23)\\
w_{ii}(p) &= 0 && \{\forall p\}
\end{aligned}$$

where δ_{ij} is the Kronecker Delta symbol. This rule keeps the amount of noise in the weights, K, approximately constant under continuous training, and it is simple to show that constraint 4.7 is satisfied since $\mathbf{W}\mathbf{v}^{(p)} = \eta \mathbf{v}^{(p)}$. ($S$ is in fact independent of η if $\eta > 0$.)

Spans

In fact this rule predicts a maximum capacity higher than that with generalised Hebbian rules. Using a non-standard measurement of "Bit Error Rate", or the probability of an initially unstable unit, [Morris & Wong 88] calculate a maximum (transient) capacity of 0.25 when the bit error rate is 0.05, together with a somewhat reduced stable capacity of 0.15. The condition that the leading diagonal of $\mathbf{W}$ is zeroed is important to prevent $\mathbf{W}$ converging on $\mathbf{I}$ (which they show occurs if the condition is removed).

Note that, although the rule looks more complicated than the Hebbian rule, it is still completely local and, computationally-speaking, updating is of the same order $O[N^2]$; in fact it only involves approximately twice as many steps.

6.5.1 Performance

Fig 6–2 shows the span of the net after training from a Tabula Rasa. The maximum number of patterns reliably retrieved is 67*, and the number does not stabilise until*

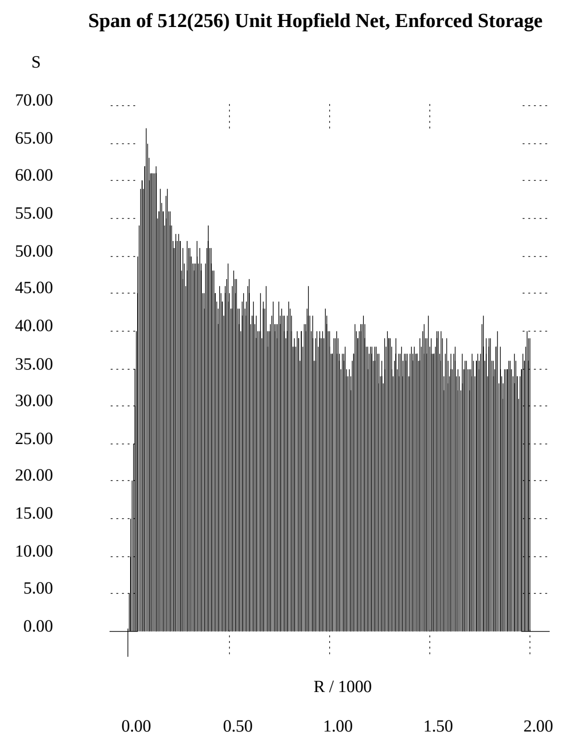


Figure 6–2: Span under Enforced Storage

about 1,000 *patterns have been trained. The average span (after initial transients have died down) is* 35.7 ± 2.2*. The mean weight value is* 0.002 ± 0.442 *when* $\eta = 10$*.*

6.6 Other Methods

Several authors have proposed more sophisticated learning rules, such as the "pseudo-inverse" method or the "eigenstructure" method. Incremental learning and forgetting regimes have also been designed for both of these [Yen & Michel 92].

However, these methods are usually very complicated and would seem biologically implausible (often violating the locality constraint). Consequently, the details are not covered here, although for reference, a summary of the four basic classes of learning rule, from [Yen & Michel 92], is shown in Table 6–2.[4]

[4]The delta rule falls into the class of "projection rules", from the fact that **W** comes to project onto the R-dimensional space spanned by the pattern vectors.

Characteristic	Hebbian	Projection	Pseudo-inverse	Eigenstructure
Net Always Stabilises	yes	yes	no	yes
Patterns always Stored	no	yes	yes	yes
Storage Capacity	$< 0.15N$	$< 0.5N$	$< 0.5N$	can be $> N$
Symmetric Weights	yes	yes	no	yes
Spurious States	yes	yes	yes	ltd
Energy Analogy	yes	yes	no	yes

Table 6–2: 4 Different Classes of Learning Rules

6.7 Summary

All the forgetting methods discussed above, given certain choices of parameters, can allow a net to function as a palimpsest with a stable span linearly proportional to the size of the net, as shown in Table 6–3, where both theoretical, St, and simulated, Sp, spans are given.

Palimpsest Scheme	Optimal Parameters	St	Sp
Bounded Weights	$\eta = \frac{3.0}{N}$, $B = \frac{1}{\sqrt{N}}$	0.04N	0.05N
Attenuated Weights	$\eta = \frac{4.1}{N}$, $\lambda = \exp\left(-\frac{8.4}{N}\right)$	0.05N	0.06N
Random Unlearning	$e = 1$, $u = 1$, $\epsilon = 1.0$	-	2
Enforced Storage	$\eta > 1$	-	0.07N

Table 6–3: Comparison of Palimpsest Schemes in Hopfield Net

However, the above analyses and simulations determine only the number of patterns that are stable states of the net when *perfectly* reproduced over the units. Of course, if $\mathbf{W} = \mathbf{I}$, then that number would be 2^N ! Consequently, it is important to examine how these schemes compare under noisy presentation of patterns - *i.e.* to elucidate the relative sizes of each desired attractor. This is examined in the next chapter.

Chapter 7

Comparison of Palimpsest Schemes

This chapter attempts a more general classification of the palimpsest schemes considered in the previous two Chapters. They are also compared, in neutral, information-theoretic terms, for auto-associative performance under noise.

7.1 Some Categorisation

The seven methods of forgetting considered in this project can be viewed as exemplars of three main categories of forgetting.

7.1.1 Weight Decay

Weight decay, as a type of forgetting process, shall be defined as gradual returning of weights to some fixed resting value, in a *non-specific* manner and as a function of the number of forgetting episodes. It is non-specific in the sense that it is independent of the nature of the patterns themselves (once forgetting parameters are fixed). Examples include the Random Resetting method in the WN, and Attentuated Weights method of the HN (resting $w_{ij} = 0$ in both cases).

Weight Ageing in the WN is another example of Weight Decay. The difference to Random Resetting is essentially between whether the probability of a switch remaining triggered over time has first- or second-order dependence on the number of forgetting episodes. When the second-order dependence in the Weight Ageing method has a very specific form (*i.e.* a step function), there is a considerable improvement in span.

Random Unlearning of patterns in the HN can also be grouped into this category. Weights effectively decay back to their mean value in a non-specific manner because the attractor to be unlearnt is randomly chosen.[1]

Weight Decay methods do not necessarily require any modification of a model's learning function, since the forgetting stage can be treated as a separate, *active* procedure in the complete training algorithm (its incorporation into a rule in the HN is simply more concise). This suggests that learning and forgetting could operate as independent processes on different timescales, *i.e.* there need not always be alternating learning and forgetting episodes, as raised in Section 6.4. The general advantages of uncoupling their timescales are not immediately apparent, and beyond the scope of this project.

7.1.2 Bounded Weights

The prototype of this category is the method of Bounded Weights in the HN. Here the weights are at no time forced to decay back to their mean value, but remain in a restricted asymptotic distribution. Restricting weights actually involves changing a linear learning function to a non-linear squashing function.[2] (Since the weights in the WN are already clipped to only one of two values, there is not really an equivalent method here.)

Forgetting due to bounded weights is also non-specific, and there is again the possibility of uncoupling learning and forgetting: *e.g.* clipping weights after several patterns have been learnt, *á la* [Amit & Fusi 92].

However, forgetting in nets with bounded weights can also be regarded as a *passive* consequence of the physical structure of a net: in the sense that it is a

[1]Indeed, a random pattern unlearning process in the WN would be formally equivalent to the random resetting method.

[2]Note that bounded weights do not have to be *clipped* to a set of discrete values: they can still be continuous.

property of the connections rather than any active process operating on them. For example, it would be possible to independently force forgetting *via* weight decay, but with bounded weights, forgetting is necessarily tied up with further learning of new patterns.

7.1.3 Pattern Interaction

This method of forgetting relies on a modified learning function where the training of the most recent pattern has a *specific* effect on existing stored patterns. In one sense, all distributed memories have some forgetting from *interaction* between patterns. However, the specificity used here refers to the forgetting being actively directed by the pattern currently being trained.

For example, consider Generalised Learning in the WN. Whereas retrieval failure under standard, non-palimpsest learning arises from a constant background noise, under the process of unlearning, the noise can be selectively modified for the current pattern.

As discussed earlier, the HN learning rule is already an example of the Generalised Hebbian Rule, so this type of effect is implicit. However, it is possible to extend the idea of specificity to the Enforced Storage case as well. Again, forgetting is a consequence of the interaction between the existing patterns and the pattern being trained, but now this interaction is further modified by how <u>much</u> the weights need to be altered to accomodate this pattern. Thus even more information is gleamed from the particular pattern being trained, and used to minimise adverse interaction with other patterns (and with the more sophisticated methods alluded to in Chapter 6, yet more information is used through comparing the new pattern with the current state of the net).

7.2 Single Unit Analysis

[Amit & Fusi 93] examine palimpsest behaviour from the point of individual threshold units, irrespective of any net architecture. By obtaining an expression for the signal-to-noise ratio for weighted sums to an individual unit after a certain number of patterns trained, they apply a constraint that the probability of an error on any unit tends to zero with increasing N, which [Weisbuch & Fogelman-Soulie 85] believe requires that the square of the signal-to-noise ratio must grow at least as $log(N)$.

With this restriction they observe a constraint only logarithmic in N on the maximum span of any net employing such units (*i.e.* less than $O[N]$), unless one of a number of various parameters is allowed to scale with N (see Section 8.1.4). Two particular parameters relevant here are the pattern coding F, which must decrease with N, or the number of clipped values of a weight, which must increase with N.

However, there seems to be a discrepancy between their theory and the results here. In examining the binary units in the WN, and a Generalised Learning scheme, they state that:

$$S = O\left[\ (\tfrac{N}{log(N)})^2\ \right] \tag{7.1}$$

which is the same order as R_c, provided:

- F decreases with N through $F \propto \frac{ln(N)}{N}$
- The unlearning parameters are related to N *via* F, by: $x + y = Fz$ (and $w = 0$; *c.f.* Section 5.3.1)

However, simulations with $M = ln(N)$ and $x = Fz$ in a 512(6) WN give very small spans ($\simeq O[10]$) - mainly due to the non-optimal pattern coding. Even the numerical analysis and simulation in Section 5.3.4, which include some cases almost satisfying the second provision (but with $M = 9$), still do not give spans of this order.

Results from $N = 256, M = 8$ *and* $N = 1024, M = 10$, *with homosynaptic unlearning as prescribed above, show spans of* 20.9 ± 3.6 *and* 130 ± 10 *respectively.*

Together with the result of $S \simeq 50.6 \pm 5.8$ *from the* $512(9)$ *net, these spans do not really show a clear scaling of* $\frac{N^2}{log(N)^2}$.

There are probably two reasons for this discrepancy. Firstly, [Amit & Fusi 93] are satisfied with the constraint that $E[H_O] \rightarrow 0$ as $N \rightarrow \infty$. This does not address suitable retrieval criteria for finite N, as are required in actual simulation estimates of S, rather than just signal-to-noise ratios. Since mathematical expressions for S have complex dependency on τ and H_L (see Appendix B), it is in fact difficult to obtain an order for the expected span.

Secondly they state:

> "In the simple case of auto-associative memory the possibility of retrieving a memory is determined by the distribution of depolarisations among the neurons in the network upon the presentation of one of the previously memorized patterns. If that distribution is such that there exists a threshold which separates the depolarization of neurons which had been active in the learned pattern from those which had been quiescent, retrieval is *in principle* possible."

(author's emphasis). Thus they simply assume adaptable thresholds for individual units can be found to separate signal from noise, based on particular distributions of weighted sums. This is not the case for the WN considered here (though this adaptability, whilst allowing better retrieval, seems unlikely to account for a differences of order of N or $log(N)$ - [Willshaw 93]).

7.3 Information Capacities

It is not necessarily advisable to compare performances of the seven different methods by direct comparison of their spans in Chapters 5 and 6. This is because of three reasons:

1. The WN results come from the hetero-associative scenario, whilst the HN can only ever function as an auto-associator.

2. The patterns used in each net have different information contents. The WN patterns always have 9 components set to 1, whereas the HN patterns have 256 such components $\underline{\text{on average}}$.

3. The critera for reliable retrieval are different: for the WN, H_L was 2, whilst for the HN, it was 7 (the latter was in order to achieve reasonably large spans and also to conform with the conventional m_L of 0.97).

To allow comparison, simulations have been carried out with the WN operating under auto-association. Then, by relativising information contents, a common reliable retrieval criterion can be chosen, and the two models compared.

Moreover, as mentioned earlier, the value of an auto-associator is only apparent when there is some uncertainty in the retrieval cue. Thus the relative metric of interest is the total amount of information, $I(n)$, given a certain amount of noise in the retrieval cues, n, that a net can *reliably store*. This can be calculated from the number patterns, $S(n)$, with information content, I_p, that can be reliably retrieved:

$$I(n) = S(n)I_p \tag{7.2}$$

Call $I(n)$ the *information capacity* of a palimpsest. (Note how this measure does not directly address the issue of how much information in the cue pattern needs to be provided in order to retrieve the full pattern. Rather the capacity here is a variable, dependent on the amount of noise in the cue.)

Information Content of Patterns

I_p can be defined by the standard Shannon Metric:

$$I_p = -N(\ qlog_2(q) - (1-q)log_2(1-q)\ ) \tag{7.3}$$

where q is the probability that a component in pattern p has one of two possible values. As noted in point 2 above, q is not strictly the same quantity for both HN and WN patterns, but the difference can be over-looked here. Then, substituting F for q: for WN patterns $I_p = 0.127N$ (with $N = 512$, $M = 9$), whilst for a HN, $I_p = N$.

Reliable Retrieval Criterion

To equate the retrieval criteria, it can be noted that $H_L = 2$ in the WN corresponds to a total error in information terms of 29% of I_p - *i.e.* this would be the minimum amount of information needed to correct the error. Then an identical total error for the HN patterns implies an H_L of 26 (using Eq 7.3).

Measure of Noise

The amount of noise, n, is a scalar measured as follows: n is the number of randomly chosen pattern components that are flipped to one of the two possible values with equal probability. Thus $n = 0$ corresponds to a perfect cue (as is the case for the results in previous chapters), whilst $n = N$ corresponds, in the average, to a cue completely uncorrelated with the intended pattern.[3]

Comparison of Information Capacities

Given these adjustments, a fairer comparison of the different palimpsest schemes (with optimal choice of parameters) is shown in Table 7–1. The figures represent the amount of information reliably retrieved per pattern component, $I(n)/N$, with maximum error of $29\% I_p$, for different values of n.

The following important conclusions arise:

- With no noise, all information capacities are of the same order, bar that with Weight Ageing and Random Unlearning of course.[4]

- The advantage of (optimal) Weight Ageing fades rapidly with small amounts of noise, *e.g.* by $n = 10$, spans have dropped by a factor of 100. This is in agreement with a general result of [Willshaw 71], which is that a net's

[3]With spin values, this means generating cues with $a_i = u_i v_i$, where $u_i = -1$ with probability $n/2N$.

[4]It is also noticeable that, in calculating $I(0)$, auto-associative spans in the WN are sometimes up to $\simeq 135\%$ bigger than hetero-associative ones.

Method	$n = 0$	$n = 5$	$n = 10$	$n = 20$	$n = 30$
Random Resetting	24.5	12.0	3.26	0.06	-
Weight Ageing	234	33.1	2.31	0.10	-
Gen. Learning	28.1	14.3	3.60	0.09	-
Bounded Weights	25.4	25.3	25.0	24.9	23.6
Atten. Weights	30.5	30.0	30.0	30.0	29.7
Random Unlearn.	2.0	2.0	2.0	1.96	1.97
Enforced Storage	36.2	30.9	27.1	17.2	8.75
	$n = 50$	$n = 100$	$n = 200$	$n = 400$	$n = N$
Random Resetting	-	-	-	-	-
Weight Ageing	-	-	-	-	-
Gen. Learning	-	-	-	-	-
Bounded Weights	24.2	23.6	20.6	8.43	0.33
Atten. Weights	29.2	28.1	25.0	9.18	0.21
Random Unlearn.	1.93	1.74	1.94	1.86	0.50
Enforced Storage	0.90	-	-	-	-

Table 7–1: Information Capacities of Palimpsest Schemes when $N = 512$

robustness to noise is effectively proportional to its “spare” capacity. Since p is over twice as big in the optimal Weight Ageing case ($p \simeq 0.44$) compared to the two other WN schemes ($p \simeq 0.17$), an increased sensitivity to noise is to be expected.

- Capacities in the HN show much more resilience to noise than in the WN. This is mainly due to the iterative updating in the recurrent HN that allows progressive correction of errors (at the expense of time).

- This is particularly true of Bounded and Attentuated Weights, where finite spans exist even when $n = 400$. This is not necessarily surprising, since the random amounts of noise mean that, given many retrieval cues, some are likely to be close enough to the desired attractor even when n is this large.

- It is not so true under Enforced Storage (ineffective when $n > 50$), where the cost of an increased number of stable pattern attractors must be a reduction in the size of their basin of attraction.

Consequently, a “take-home” lesson from this section might be: if the interest is in correcting a small number of spurious errors in a given cue, Weight Ageing in the Willshaw Net has the greatest information capacity. However, if the interest is retrieving patterns from very noisy data (as would seem more useful), then Attenuated Weights in the HN is best.

A Word on other Models and Architectures

A slightly different perspective is taken by [Amari 88], who considers the "noise-reduction property" of associative nets with L layers, where layer l feeds forward to layer $l+1$. This property is a measure of how much the overlap, between the cue on layer 1 and target pattern, improves by the last update (*e.g.* on layer L), as a function of the loading, α, of the net.

Recurrent nets are obtained when layer L feeds back into layer 1. Then a HN, with parallel update, corresponds to the case when $L = 1$. The interesting result of Amari's neurodynamical analysis is that noise-reduction in recurrent nets is "better" when $L = 2$ than when $L = 1$. Thus feeding output back to input in a WN may improve the information capacity of WN palimpsest schemes under noise.

However, when $L \geq 3$, the advantage of recurrency itself decreases. Thus feedforward architectures with $L > 2$, and parallel update, buy better and better noise-reducing ability at the cost of an increased number of connections.

Chapter 8

Perspectives

This chapter gives brief consideration to some physiological, psychological and implementational aspects of the palimpsest schemes studied.

8.1 Physiological Plausibility

8.1.1 Timescales

The average firing rate of neurons in the brain places an upper limit on the number of calculations they can usefully perform in one second. The Feldmann 100-step rule estimates this number to be of the order of 100 basic operations, which is much less than is possible with the clock-rate of conventional computers. This is often taken as an argument for the importance of parallel computation in the brain.[1]

Some have taken this as prohibitive of iterative updating techniques, *e.g.* [Sejnowski 86], where only one neuron updates in successive time-steps. For a net of thousands of units, thousands of updates (as in the HN) would be needed to reach a stable state, which would be too slow for the sort of computations that animals appear to be capable of. There is no way that a human brain with at least

[1]Though others have observed that, if information is encoded in firing probabilities rather than rates, there is not necessarily such a strong constraint.

10^9 neurons, could function as a single, huge HN ! This is often used in support of parallel update, such as is common in feed-forward nets.

Another issue is the relative timescales of updating and learning. If there is no explicit distinction made in the brain between learning and retrieving episodes, learning must be continuous. Then, for a net to learn desired patterns or states, but not states intermediate in the process of retrieval, the learning process must be relatively slow compared to the updating process (or desired states must be maintained by external stimuli for a long time relative to retrieval times). This again favours fast (parallel) update.

8.1.2 Connectivity and Coding

Although the average "fan-in" of neurons in animal brains is large ($10^4 - 10^6$), the connectivity is still sparse compared to the number of neurons. Sparse connectivity has been studied in many papers, with the main result being that the decrease in capacity is actually accompanied by an increase in *efficiency* (see Section 8.3). There is no reason to think that this would not also be the case for the palimpsests considered here, so sparse connectivity has not been studied here.

Many physiologists believe that the brain also employs sparse coding of stimuli, which may be modulated by more general activation-inhibiting processes (that could dynamically adjust the threshold with M_I in the WN for example). This is again associated with increased storage efficiency (in spite of the smaller amount of information in individual patterns). Thus the sparsity of coding in the WN may be justified on these grounds.

8.1.3 Synapses

Realistically, synapses must have a limit to their efficacy and their learning function is almost certain to be non-linear. Synaptic efficacies are also likely to be continous rather than discrete or binary as in the WN (even though clipping of synapses leads to greater efficiency in artificial neural nets [Hopfield 82]).

The exact operation of synapses would seem to be very complex, so most learning functions are considerable abstractions. Needless to say, Hebbian reinforcement is a gross simplification. However, as a simplification, it would seem more plausible than any delta-rule form of learning (the delta-rule may be more appropriate at a higher level of description than the neuronal level). As discussed in section 5.3.1, LTD has only been confidently observed in the Cerebellum, and has not been observed yet in the classic "home" of LTP, the Hippocampus. Also, it would seem too early to make conclusions on the presence of synaptic decay over time. There is sometimes a gradual reduction in LTP over a period of days, but whether this is the only type of decay is an open question.

There is a fundamental tenet in neurophysiology [Crick & Asanuma 86] known as Dale's Law. This is an empirical finding, that synapses can be either excitatory or inhibitory, but can never change from being one type to being the other. The law is not respected in any of the HN palimpsest schemes considered here. However, [Nadal *et al* 86] did find that adopting a sign constraint on weights (such that either $-A \leq w_{ij} \leq 0$ or $0 \leq w_{ij} \leq A$) does not qualitatively affect the performance of a palimpsest with Bounded Weights. (Another common response is to dismiss the issue by noting the equivalence of having two connections between units, one which can only be negatively weighted and the other only positively weighted).

8.1.4 Optimality

The optimal choice of forgetting parameters in all palimpsest schemes considered here would seem to require knowledge of N and F.

It might be plausible that the numerical parameters in learning functions have evolved to near-optimal values, but given that there is much structure in the animal brain, with different codings and neuron concentrations, it seems implausible that the variables such as r, $w - z$, B, λ and particularly A_o,[2] have evolved optimally for different areas of the brain. Moreover, knowledge of N and F would not seem to be locally accessible, so neither does it seem likely that the dependencies can be explicitly determined (though one could imagine the variables being partially determined by the number synapses impinging the same neuron and/or some

[2]This is why the dependency of S on A_o has been "diluted" by the D parameter in the more biologically-plausible sigmoidal $r(A)$ functions.

estimation of the stimuli coding from sampling the average activity of a neuron over time). Thus another point in favour of one palimpsest scheme over another would be a weaker dependence on parameters such as N and F.

8.1.5 Summary

As mentioned in the introduction, current neural nets are a far abstraction from reality. This caveat in mind, the tentative conclusion of this section would be that Generalised Learning and/or Bounded (but continuous) Weights would seem to be the most biologically plausible forgetting methods. This is because there is already some evidence for the former, and the latter is very likely true of real synapses. This is not to say that some form of Weight Decay over time is implausible; simply there is no conclusive evidence for it yet.

As for the different models, the architecture of the Willshaw Net would seem more plausible than that of the Hopfield Net, particularly if sparsely connected; the feedforward and parallel update features appearing most important.[3] For a good review of architectures for auto-associative memory, see [Treves & Rolls 91].

8.2 Psychological Comparisons

The categories of palimpsest schemes attempted in the previous Chapter already have analogies in psychological theories of forgetting. The debate between temporal decay theories (*c.f.* Weight Decay) and inteference theories (*c.f.* Pattern Interaction) has been raging for decades [Baddeley 86]. With the development of neural networks, support was gained for interference theories, since this is the reason for normal (catastrophic) failure of such distributed memories. However, as has been shown by the methods of Weight Ageing and Attenuated Weights, decay alone can also produce effective forgetting for short-term associative memories.

[3]This is not denying the utility of the Hopfield Model at a higher level of description of associative memory.

However work on forgetting methods in neural networks may have exposed some more subtle distinctions. One is that between specific and non-specific methods, as discussed in section 7.1. Another is that between active and passive forms of interference. For example, the forgetting induced by bounded or clipped weights is a passive form of interference, more akin to the notion of *masking*.

One important and robust result from experiments on human short-term memory is a "saucer-shaped" serial order curve. These curves not only show a *recency* effect (the advantage of more recent training of patterns, as is clearly observed in all schemes considered here), but also a *primacy* effect, where the first few items of the training sets also have an increased chance of recall. None of the palimpsest schemes considered show this.

There have been several mechanisms suggested by which primacy can arise, the most common being the increased "saliency" of the first few items (perhaps a higher initial η variable ?). [Wong *et al* 91] suggest a Bounded Weights model, with differing probabilities of strengthening and weakening weights (when the probability of strengthening weights is greater, primacy can be exhibited in addition to recency). [Burgess *et al* 91] obtain both by combining absorbing weight bounds, *á la* [Peretto 86], with Weight Attentuation. Alternatively, there may be associations learnt not simply between the items themselves, but also the "experimental context" [Burgess & Hitch 92], with stronger context-item associations for earlier items in a list. This obviously needs some consideration of the transient effects of individual learning occasions (*e.g.* each new occasion implicating a Tabula Rasa ?). Finally, [Nadal *et al* 87] give some speculative comments about increased performance through repeated learning, which is observed both in people and in their palimpsests, and some psychological analogies to transients seen in Tabula Rasa training.

Thus from the psychological perspective, care must be taken in comparing forgetting methods in general, and forgetting methods perhaps unique to short-term memories. For a very simple model of short- and long-term memory with both temporal decay and interference, see [Gardner-Medwin 89].

8.2.1 Summary

As apparent, there is much more work needed before artificial neural network models can account for the wealth of experimental data obtained from psychological investigation. Only a tiny number of observed phenomena have been mentioned above - although recent research does seem to be heading in this direction. However, a major issue in connectionist modelling, not touched upon here, is the amount of structure provided in the net (the so-called "modularity" debate). Since there is undoubtedly much structure in brains, the "flow" of processing must be relevant the problem of forgetting, *e.g.* whether loops exist which periodically refresh decaying memories.

8.3 Implementational Perspective

Section 7.3 shows the information capacity of the seven palimpsest schemes under different amounts of noise. Another useful metric is their *information efficiency*, ξ, which is the ratio of their capacity to the amount of information needed to specify the net itself (*e.g.* the amount of data needed to implement the net on a conventional computer). This is obviously an important consideration for application of neural network palimpsests.

8.3.1 Implementation Requirements

Willshaw Net

Since weights are binary, the number of bits to specify $\mathbf{W}$ is N^2. This is all that is needed for the Random Resetting and Generalised Learning scenarios (storage of activations and other parameters being negligible in comparison).

However, the Weight Ageing method requires the additional storage of the age of each weight. The upper bound of each age is the integer A_o, and a realistic maximum value of this parameter is $\frac{N^2}{M^2}$ (since larger values are unlikely to produce a useful palimpsest). Thus let the extra amount of data required be $N^2 log_2(\frac{N^2}{M^2})$.

Hopfield Net

Here the weights can be real-valued, but the palimpsest schemes effectively impose a maximum absolute value.

For Bounded Weights, assume that the bounds $\pm B$ give a range of $2B/\eta$ possible integer values for w_{ij}. Then $\mathbf{W}$ requires $N^2 log_2(2\frac{B}{\eta})$ bits of information.

For Attenuated Weights, an upper limit on $|w_{ij}|$ is $\frac{\lambda\eta}{1-\lambda}$. The w_{ij}'s can have any real number in this range, but assume that they are stored only to 2 significant figures[4]. Then $\mathbf{W}$ requires $N^2 log_2(200\frac{\lambda\eta}{1-\lambda})$ bits of information.

For Enforced Storage, analysis is difficult. However, an estimate can be obtained from practice. In a 512(256) net, the experimental variance of the weights after extensive training can be used as an estimate for a $\simeq 95\%$ probability of each weight being under an absolute value of 0.884 (when $\eta = 10$ in Section 6.5.1). If it is assumed that the other 5% are clipped with negligible effect on S, the learning rule implies a range of possible values of $2\frac{0.884N}{\eta}$, from which storage cost can be determined.

8.3.2 Comparison of Information Efficiences

Table 8–1 summarises the approximate information efficiences ($\times 10^{-3}$) of six of the palimpsest schemes (Random Unlearning is not shown since its information capacity is so small), under various conditions of noise.

The following important conclusions arise:

- With no noise, information efficiencies are all roughly of the same order, *i.e.* the advantage in $I(n)$ under Weight Ageing is compensated by a greater implementation cost.
- However ξ for WN palimpsest schemes is generally higher than for HN schemes, provided there is little noise. This is because the simple binary na-

[4]This choice of accuracy (clipping) may be a very poor one - consideration of necessary accuracies is beyond the scope of this simple comparison.

Method	Bits/512^2	$n = 0$	$n = 5$	$n = 10$	$n = 20$
Random Resetting	1.0	48	23	6.4	0.1
Weight Ageing	13	36	5.0	0.4	-
Gen. Learning	1.0	55	28	7.0	0.2
Bounded Weights	3.9	13	13	13	12
Atten. Weights	6.6	9.0	8.9	8.9	8.9
Enforced Storage	5.5	13	11	9.6	6.1
	$n = 30$	$n = 50$	$n = 100$	$n = 200$	$n = 400$
Random Resetting	-	-	-	-	-
Weight Ageing	-	-	-	-	-
Gen. Learning	-	-	-	-	-
Bounded Weights	12	12	12	10	4.2
Atten. Weights	8.8	8.6	8.3	7.4	2.7
Enforced Storage	3.1	0.3	-	-	-

Table 8–1: Information Efficiencies of Palimpsest Schemes when $N = 512$

ture of the WN makes it more efficient to implement (this has been observed in many other papers). Note again that the high efficiency of the WN is contingent on patterns being sparsely coded. Interestingly, [Kohring 90] describes a method of converting single, densely coded patterns into a number of sparsely coded patterns, and retrieving these in limit-cycles in a modified Willshaw Net.

- Maximum values of ξ are still only $O[0.01]$, *i.e.* $\ll 1$. Thus the price of effective forgetting for a short-term memory is (obviously) a reduction in information efficiency, *c.f.* maximum efficiency of the WN of $O[1]$.

- Obtaining an upper bound on w_{ij} from experimental results seems a satisfactory method, since it provides an acceptable estimation when applied to Bounded Weights as well as Enforced Storage. However, when applied to Attenuated Weights, it gives an upper bound $\simeq 10$ times smaller than the bound used in the table above (obviously the theoretical upper limit, obtained from the summation of a geometrical progression, is never approached in practice). This may explain why the efficiency of Attenuated Weights is markedly reduced over the other HN schemes.

8.3.3 Summary

Consequently, a "take-home" lesson from this section might be: given low noise levels, Generalised Learning is a good option, whereas, given potentially high noise levels, Bounded Weights is generally the most efficient option in information terms.

Finally, from the application viewpoint, it has been observed that neural networks are only more *computationally* efficient than conventional methods when the number of stored patterns per unit is greater than the number of connections per unit. In this case, none of the above methods are anywhere near as efficient as a simple FILO or FIFO stack of limited capacity. Of course, what the latter lack is any content-addressibility: retrieval of information must be a separate and error-free procedure.

Chapter 9

Discussion

9.1 Conclusion

It is best to present conclusions in relation to the original aims.

Identification

The author believes this project to have examined <u>all</u> the palimpsest schemes suggested in the literature to date that satisfy the locality constraint on learning functions (which can be argued as being one of the most important features of cognitive parallel, distributed processing). In addition, as far as is known, the methods of Ageing Weights, Generalised Learning and Random Unlearning have not been employed before to create neural network palimpsests.[1]

Further, if some ideas have been overlooked, *e.g.* in other types of neural network models, they may well still fall into one of the three main categories identified in the classification attempted here. Chapter 7 then provides a guide to accompany any journeys into into less well-charted neural network territory.

All seven of the palimpsest schemes considered in this project: Random Resetting, Ageing Weights, Generalised Learning, Bounded Weights, Attenuated Weights, Random Unlearning and Enforced Storage, entail forgetting of memories over and above the normal interference found in artificial neural networks. More specifically, the forgetting actually prevents catastrophic interference and allows

[1]The [Amit & Fusi 93] paper appeared subsequent to the work in section 5.3.1.

palimpsest-like behaviour, where the associative memory maintains its ability to (temporarily) store new patterns indefinitely.

Implementation

Results from each scheme have been presented from simulations of Willshaw and Hopfield Nets with 512 units (and 512×512 connections). The basic results of interest are the spans, their stability and the "sigmoidal" serial order curves, that show priviliged storage of recent memories and gradual forgetting of older memories.

Comparison

The cost of indefinite learning is a reduction in capacity compared to the maximum under normal (restricted) learning. For schemes in the HN, this reduction is generally about 50% (except with Random Unlearning, which is not so effective). In the WN, the price is greater, with spans about a tenth as large (except with Ageing Weights, in which extra information can avoid any reduction at all).

When the span is close to maximum capacity, serial order curves become very sharp, approximating step functions. This would seem necessary if the possibility of catastrophic failure is to be discounted. For small spans on the other hand, curves are typically characterised by exponentially decelerated forgetting.

The differences between the optimum spans of palimpsest schemes are mainly due to the different neural network models to which they are tied. Once auto-associative capacities are compared in information theoretic terms, differences are diminished markedly. They only remerge when increasing noise is introduced into retrieval cues, whence the iterative nature of the HN maintains capacities longer.

Generalisation

A necessary condition of palimpsest behaviour is an asymptotic (bounded) distribution of weight values. It is not sufficient however. (Consider the WN, where $p = 1$ is actually the asymptotic distribution that epitomises catastrophic failure.)

The distribution must further be such that new learning can always take place, *i.e.* learning must be able to change at least some weight values.

Asymptotic distribution can be achieved by balancing potentiation and depression of weights[2]: in a manner specific to the current pattern being trained (in Pattern Interaction methods) or in a non-specific manner (in Weight Decay methods). Alternatively, potentiation may be restricted, limiting the space of possible weight values (in Bounded Weight methods). Further distinction between these three categories of palimpsest scheme are highlighted in Section 7.1.

Analysis

Mathematical analysis helps the understanding of palimpsest behaviour and allows prediction (and optimisation) of spans. It has been attempted, or referred to in cited papers, for all palimpsest schemes except perhaps Random Unlearning (due to time constraints). However, there are points where complexity of analysis means that it must give way to numerical methods of prediction, *e.g.* with Random Resetting and Generalised Learning.

The various analyses have been of three main quantities: probability of switches being triggered for signal and noise, in the WN, and signal-to-noise ratio of local fields and variability of weights, in the HN. Furthermore, the general classification of neural networks in Chapter 2 introduced variables common to both the WN and the HN. Though this has meant that some "conventional" variables in both models have been replaced, its value has been to allow analytical comparison of the WN and HN, and hence ease the process of generalisation across models. Few authors have provided such a common framework, in which palimpsest ideas can be expressed.

[2]Potentiation and depression, as used here, refer to increases or decreases in the *magnitude* of weights.

Perspectives

In addition to information capacity and efficiency, the palimpsest schemes also differ from other perspectives. The main conclusions of Chapter 8 are:

Physiology The brain's methods of forgetting may well include limited ranges of synaptic efficacies and LTD, though there is no conclusive evidence for synaptic decay over short time intervals. However, there is much yet to learn about the physiology of the brain.

Psychology As psychological models, neural networks are only beginning to be applied to scientific knowledge over 40 years old. However, they do offer much hope, since their style of computation is obviously more brain-like than conventional computers: they have replaced the Von Neumann machine as the general, paradigmatic analogy. Recent work on palimpsests then represents the first few steps of applying them to more specific cognitive components such as short-term memory - though again many more steps are needed before these models reveal significant scientific value by actually making predictions for further psychological experimentation.

Application In a situation where the need is simply to store the last X items of a continuous stream of inputs, the most efficient way would still seem to be conventional addressing methods (where the latest item replaces the oldest through a in-built procedure). Much efficiency must be sacrificed for content-addressibility and damage-resistance. Then, for hetero-association, WN schemes such as Generalised Learning are most efficient. On the other hand, for auto-association, the only realistic choice is to use a HN and a Bounded or Attenuated Weights scheme.

9.2 Extensions

One major confinement of this investigation has been to assume uncorrelated sequences of randomly-generated patterns (*c.f.* Section 5.3.2). Correlation can, as mentioned in the introduction, have both spatial and temporal aspects. Both of these merit much more attention than is given in the literature on associative memories.[3] Correlations will have major effects on span and serial order profiles. In fact, uncorrelated stimuli would seem to be rare in our environment - it may be that the reason that do not notice so much forgetting of long-term memories is that correlations preserve commonalities between memories; economy is achieved through classification (*e.g.* prototypical or "generic" memories).

Another possible extension would be to examine other neural network models, for example, the feedforward nets with several layers as mentioned in Section 7.3. Forgetting could also be studied in relation to continuous rather than discrete activations (themselves possibly decaying over time). Another popular area of interest is in stochastic update rules, as in Boltzmann machines, and stochastic clipping of weights, as has been mentioned in Section 6.2. At several points the possibility of uncoupling learning and (active) forgetting procedures has been suggested. This too may warrant investigation.

As stated at the outset, net performance under different net sizes or different pattern codings has not been studied. Though mathematical expressions for span have involved these variables, validation has only been with a fixed value for each. Varying N and F can be easily achieved with the current Simulators, but it has not been attempted in this project due to time constraints (particularly since the analyses are typically based on assuming large values of N, and this involves much computation). Particularly dramatic might be failure of the WN as the sparse coding limit is broken. Also, a span-optimal information capacity and coding might be derivable and testable for the WN. Another interesting possibility would

[3]Indeed, the papers that do observe this limitation, invariably quote it as an extension as well !

be to test further the effects of the large N assumptions, examining discrepancies for small ($N < 512$) nets.

Finally, if the locality constraint is removed, investigation of the information capacity of more complicated learning and forgetting methods in the HN might be achieved (such as the Eigenstructure method). This might be appealing from the application perspective, since these methods usually possess many additional, desireable properties (such as control of the number of spurious attractors).

9.2.1 Summary

This report provides, in a uniform terminology, a comprehensive overview of current research in short-term associative memories, together with substantial work on some novel ideas not immediately apparent from the literature. Short-term neural networks, since their birth and "palimpsest" christening in the physics community, are now becoming of increasing interest to cognitive scientists as well.

Bibliography

[Amari 88] S Amari. Associative memory and its statistical neurodynamical analysis. In H Haken, editor, *Neural and synergetic computers*, pages 85–99. Springer-Verlag, 1988.

[Amit & Fusi 92] D Amit and S Fusi. Constraints on learning in dynamic synapses. *Network*, 3, 1992.

[Amit & Fusi 93] D Amit and S Fusi. Learning in neural networks with material synapses. PREPRINT: Neural Computation, June 1993.

[Amit *et al* 85] D Amit, H Gutfreund, and H Sompolinksy. Statistical mechanics of neural networks near saturation. *Physical Review*, 1985.

[Baddeley 86] A Baddeley. *Working Memory.* Oxford: Claredon Press, 1986.

[Buckingham & Willshaw 92] J Buckingham and D Willshaw. Performance characteristics of the associative net. *Network*, 3, 1992.

[Burgess & Hitch 92] N Burgess and G Hitch. Toward a network model of the articulatory loop. *Journal of memory and language*, 31, 1992.

[Burgess *et al* 91] N Burgess, J Shapiro, and M Moore. Neural network models of list learning. *Network*, 2, 1991.

[Crick & Asanuma 86] F Crick and C Asanuma. Certain aspects of the anatomy and physiology of the cerebral cortex. In D Rumelhart and J McClelland, editors, *Parallel Distributed Processing: Volume II*, chapter 20, pages 333–371. MIT Press, 1986.

[Dayan & Sejnowski 93] P Dayan and T Sejnowski. The variance of covariance rules for associative matrix memories and reinforcement learning. *Neural Computation*, 5, 1993.

[Dayan & Willshaw 91] P Dayan and D Willshaw. Optimising synaptic learning rules in linear associative memories. *Biological Cybernetics*, 65, 1991.

[Derrida & Nadal 87] B Derrida and J Nadal. Learning and forgetting on asymmetric diluted neural networks. *Journal of Statistical Physics*, 49(6), 1987.

[Derrida *et al* 87] B Derrida, E Gardner, and A Zippelius. Partially-connected neural networks with asymmetric weights. *Europhysics Letters*, 4(167), 1987.

[Gardner-Medwin 89] A Gardner-Medwin. Doubly modifiable synapses: a model of short and long term auto-associative memory. *Proc. Royal Society, London*, 238, 1989.

[Geszti & Pazmandi 87] T Geszti and F Pazmandi. Learning within bounds and dream sleep. *J. Phys. A: Maths. Gen.*, 20, 1987.

[Gordon 87] M Gordon. Memory capacity of neural networks learning within bounds. *Journal de Physique*, 48, 1987.

[Hertz *et al* 91] J Hertz, A Krogh, and R Palmer. *Introduction to the theory of neural computation.* Addison-Wesley Publishing Company, 1991.

[Hopfield 82] J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79, 1982.

[Hopfield *et al* 83] J Hopfield, D Feinstein, and R Palmer. Unlearning has a stabilising effect in collective memories. *Nature*, 304, July 1983.

[Horn & Usher 88] D Horn and M Usher. Capacities of multiconnected memory models. *Journal of Physics, France*, 49, 1988.

[Kleinfeld & Pendergraft 87] D Kleinfeld and D Pendergraft. Unlearning increases the storage capacity of content addressable memories. *Biophysical Journal*, 51, January 1987.

[Kohring 90] G Kohring. Performance enhancement of willshaw type networks through the use of limit cycles. *J. Phys. France*, 51(21), November 1990.

[Mezard *et al* 86] M Mezard, J Nadal, and G Toulouse. Solvable models of working memories. *Journal de Physique*, 47, 1986.

[Morris & Wong 88] R Morris and W Wong. A short-term neural network memory. *Siam Journal of Computing*, 17(6), December 1988.

[Nadal 86] J Nadal. Correction/addition. *Europhysics Letters*, 2(4), 1986.

[Nadal *et al* 86] J Nadal, G Toulouse, J Changeux, and S Dehaene. Networks of formal neurons and memory palimpsests. *Europhysics Letters*, 1(10), May 1986.

[Nadal *et al* 87] J Nadal, G Toulouse, and M Mezard. Neural networks: learning and forgetting. In R Cotterill, editor, *Computer Simulation in Brain Science*, pages 221–231. Cambridge University Press, 1987.

[Palm 88] G Palm. Local synaptic rules with maximal information storage capacity. In H Haken, editor, *Neural and Synergetic Computers*, pages 100–111. Springer Series, 1988.

[Palm 92] G Palm. On the information storage capacity of local learning rules. *Neural Computation*, 4, 1992.

[Parga 89] N Parga. Neural networks as models of associative memories. *Computer Physics Communications*, 55, 1989.

[Parisi 86] G Parisi. A memory which forgets. *J. Phys. A: Maths. Gen.*, 19, 1986.

[Peretto 86] P Peretto. Neural networks: learning and forgetting. In R Cotterill, editor, *Computer Simulation in Brain Science*, pages 98–127. Cambridge University Press, 1986.

[Personnaz *et al* 86] L Personnaz, I Guyon, and G Dreyfus. Neural network design for efficient information retrieval. In E Bienenstock, editor, *Disordered Systems and Biological Organisation*, pages 227–231. Springer-Verlag, 1986.

[Sejnowski 86]	T Sejnowski. Open questions about computation in the cerebral cortex. In D Rumelhart and J McClelland, editors, *Parallel Distributed Processing: Volume II*, chapter 21, pages 372–389. MIT Press, 1986.
[Shinomoto 87]	S Shinomoto. Memory maintenance in neural networks. *J. Phys. A: Maths. Gen.*, 20, 1987.
[Sompolinsky 86]	H Sompolinsky. Neural networks with non-linear synapses and a static noise. *Physical Review A*, 34(3), 1986.
[Stanton & Sejnowski 89]	P Stanton and T Sejnowski. Associative long-term depression in the hippocampus. *Nature*, 339, 1989.
[Stone 86]	G Stone. An analysis of the delta rule and the learning of statistical associations. In D Rumelhart and J McClelland, editors, *Parallel Distributed Processing: Volume I*, chapter 11, pages 444–459. MIT Press, 1986.
[Treves & Rolls 91]	A Treves and E Rolls. What determines the capacity of autoassociative memories in the brain? *Network*, 2, 1991.
[vanHemmen *et al* 88]	J van Hemmen, G Keller, and R Kuhn. Forgetful memories. *Europhysics Letters*, 5(7), April 1988.
[Weisbuch & Fogelman-Soulie 85]	Weisbuch and F Fogelman-Soulie. Scaling laws for the attractors of hopfield networks. *J. Physique. Lett.*, 2, 1985.
[Willshaw & Dayan 90]	D Willshaw and P Dayan. Optimal plasticity from matrix memories: what goes up must come down. *Neural Computation*, 2, 1990.

[Willshaw & Morris 89]	D Willshaw and R Morris. Must what goes up come down ? *Nature*, 339, 1989.
[Willshaw 71]	D. J. Willshaw. *Models of distributed associative memory.* Unpublished PhD thesis, University of Edinburgh, 1971.
[Willshaw 93]	D Willshaw, 1993. Personal Communication.
[Willshaw *et al* 69]	D Willshaw, O Buneman, and H Longuet-Higgins. Non-holographic associative memory. *Nature*, 222, 1969.
[Wong *et al* 91]	K Wong, P Kahn, and D Sherrington. A neural network model of working memory exhibiting primacy and recency. *J. Phys. A: Maths. Gen.*, 24, 1991.
[Yen & Michel 92]	G Yen and A Michel. A learning and forgetting algorithm in associative memories: The eigen-structure method. *IEEE Trans. Circuits and Systems: Volume II*, 39(4), April 1992.

Appendix A

Glossary

A.1 General Conventions

- Lower-case, bold symbols represent vectors.
- Upper-case, bold symbols represent matrices. $\mathbf{I}$ is the identity matrix.
- Lower-case, Greek symbols always refer to numerical constants.
- Upper-case, Greek symbols ϕ and ψ denote arbitrary functions.
- Subscripts refer to vector components.
- Superscripts individuate set members.
- $\rightarrow$ denotes a value change from one timestep to the next.
- A net abbreviation $X(Y)$ means X units and Y expected pattern components with a binary value of 1.
- $E[X]$ denotes the expected value of X.
- $O[X]$ denotes the order of X.
- A numerical result $x \pm y$ refers to a mean of x with standard deviation y (y is NOT the standard error in this report).

Variable	Description
L	Number of layers in a net
N_l	Number of units in lth layer
R	Number of patterns trained
$a_i^{(l)}$	Activation of ith unit in lth layer
$\mathbf{a}^{(l)}$	Activation vector of lth layer
w_{ij}	Weight of connection from unit j, to unit i
$\mathbf{W}$	Weight Matrix
$v_i^{(l)}$	ith pattern component for lth layer
$\mathbf{v}^{(l)}$	Pattern vector for lth layer
$n_i^{(l)}$	ith noise component for lth layer
$\mathbf{n}^{(l)}$	Noise vector for lth layer
τ_i	The threshold for unit i
F_p	The pattern coding for pattern p
$Q(p)$	Probability of reliable retrieval of pattern p
W	Window of patterns to be retrieved
f	Update function for units
g	Learning function for connections
H	Hamming distance between 2 vectors
H_O	Output Hamming Distance
H_L	Output Hamming Limit for Reliable Retrieval
S	Number of patterns stored (*e.g.* span)
μ	A mean
σ	A variance
ρ	Signal/Noise ratio
I	Information content/capacity
ξ	Information efficiency

Table A–1: General Variables

A.2 Variables

A summary of the variables used in this report is shown in Tables A–1 to A–3.

WN Parameters	Description
N_I	Number of units in Input Layer
N_O	Number of units in Output Layer
M_I	Number of 1 components in Input patterns
M_O	Number of 1 components in Output patterns
p	Probability that a random switch is triggered (the loading density)
p_c	Capacity-optimal loading density (=0.5)
p_s	Probability that a signal switch is triggered
p_n	Probability that a noise switch is triggered
R_c	Number of patterns trained for $p = p_c$
R_s	The expected survival time for associations
$P(\tau)$	Probability of weighted sum $\geq \tau$
A	Age of a switch
A_o	Critical age of a switch
D	Sharpness of a sigmoid
r	Probability of resetting any switch
w	Probability of keinosynaptic resetting
x	Probability of homosynaptic resetting
y	Probability of heterosynaptic resetting
z	Probability of triggering a switch

Table A–2: WN Variables

HN Parameters	Description
$\mathbf{s}$	State of HN at any one time
$\hat{\mathbf{s}}$	A stable state of a HN
η	Learning constant
m	Measure of overlap
m_o	Overlap between stable state and pattern
m_L	Overlap Limit for Reliable Retrieval
$\mathbf{h}(\mathbf{s})$	Local field for units in state $\mathbf{s}$
$E(\mathbf{s})$	The energy of state $\mathbf{s}$
α	The standard capacity of a HN
K	Noise, or variability, in weights
B	Bound on weights
λ	Attenuation of weights
u	Number of unlearning trials
e	Number of learning trials before unlearning
ϵ	Unlearning parameter

Table A–3: HN Variables

Appendix B

Signal-to-Noise Analysis and the Willshaw Net Palimpsest

In Section 5.1.2, a general version of the Random Resetting palimpsest scheme in the WN was introduced. However, mathematical expression and maximisation of expected survival time, R_s, (and hence span) is difficult to ascertain because of the complex form of the reliable retrieval constraint when $\tau < M$ (Eq 5.20). This appendix considers some approximations which help such analysis.

B.1 Signal-to-Noise Ratios

Now the desire is to maximise R_s with respect to r, z, τ. The first two of these factors determine the shape and relative positions of two distributions of weighted sums: for the signal and for the noise. For greatest R_s, the discriminability of these distributions must be maximised (see Fig B–1). Once it is maximised, an optimal τ can be chosen.

Making the unit usage assumption, the distributions of weighted sums for an output unit will follow a binomial distribution. The two distributions of interest are the signal distribution, $\phi(M, p_s)$, when the output unit should be activated, and the noise distribution, $\phi(M, p_n)$, when the unit should remain quiescent.

Discriminability is often measured in terms of a signal-to-noise ratio. If the distributions are Gaussian with means and variances of μ_s, μ_n and σ_s, σ_n respectively,

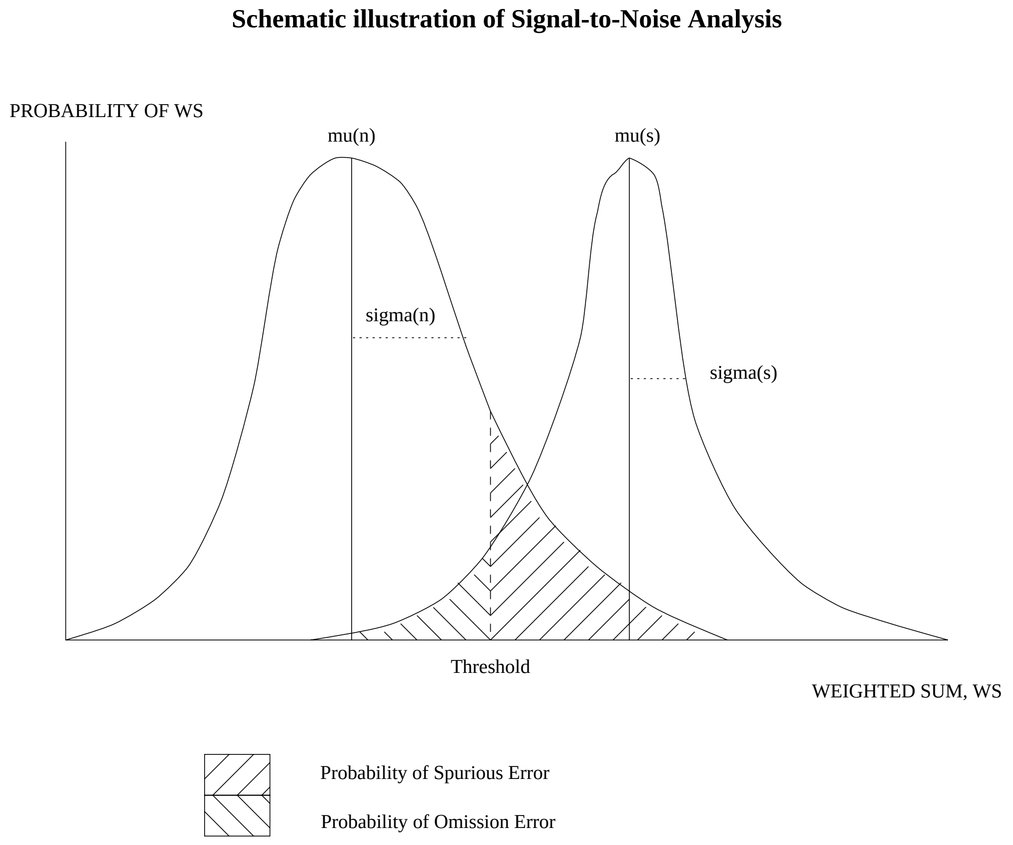

Schematic illustration of Signal-to-Noise Analysis
PROBABILITY OF WS
mu(n)
mu(s)
sigma(n)
sigma(s)
Threshold
WEIGHTED SUM, WS
Probability of Spurious Error
Probability of Omission Error

Figure B–1: The Signal-to-Noise Ratio

then the signal-to-noise ratio, ρ, is defined as:

$$\rho = \frac{(\mu_s - \mu_n)^2}{\frac{1}{2}(\sigma_s + \sigma_n)} \tag{B.1}$$

(Note how this measure is independent of any threshold used to distinguish non-signals from signals.)

For large M, the Gaussian can be approximated by a Binomial distribution. With binomial distributions, $\mu_s = Mp_s(R)$, $\sigma_s = Mp_s(R)(1 - p_s(R))$, $\mu_n = Mp_n$ and $\sigma_n = Mp_n(1 - p_n)$. Letting $k = (1 - r - zF^2)$, the ratio becomes:

$$\rho(r, z, R) = \frac{2Mrk^{2R}z}{(2F^2 + (r - zF^2)k^R - rk^{2R}z)} \tag{B.2}$$

From this equation, it can be seen that the maximal ρ will arise when $z \to 1$. (Obviously, for finite r, ρ is also maximised as $R \to 0$.)

Taking $z = 1$, assuming $R > 0$ and $r \ll 1$, $F^2 \ll 1$, then $k^R \simeq (1 - R(r + F^2))$, and ρ becomes:

$$\rho(R, r) = \frac{2Mr(1 - 2R(r + F^2))}{R(r + F^2)^2} \tag{B.3}$$

As R gets large, ρ behaves like $\frac{r}{r+F^2}$: then ρ tends to 0 as $r \to 0$, since the distributions become superimposed - consequently r must remain small but finite.

B.2 Span

When r is small and R is big, $\rho(R, r)$ could in principle be maximised subject to the constraint in Eq 5.20. Then the optimal $S(= R_s)$ could be expressed in terms of r (*i.e.* in order to obtain an estimation of S, knowledge of the threshold is required).

However, in practice, the complex form of the constraint makes such a Lagrangian differentiation very difficult. It is possible to make some small inroads by considering some good choice of thresholds and approximations of Eq 5.20.

Thresholding

Generally, a good choice of τ is given by the point of intersection of the signal and noise distributions. This is optimal in the sense that, for one output unit, if τ increases or decreases beyond this point, the combined probability of an error, omission <u>or</u> spurious, must increase. (It is far from perfect, because here there are more units in danger of emitting spurious errors than omission errors.) It is given by:

$$p_n^\tau (1 - p_n)^{M-\tau} = p_s(R)^\tau (1 - p_s(R))^{M-\tau} \tag{B.4}$$

Solving for τ gives:

$$\tau(R, r) \simeq \frac{M log(R(r + F^2))}{log(\frac{F^2 R}{1-Rr})} \tag{B.5}$$

Tractable Constraints

Slightly stronger, but more tractable, constraints than Eq 5.20 can be imposed when the retrieval limit consists of X expected omission errors <u>and</u> Y expected spurious errors, where X, Y are integers and sum to H_L, *i.e.* $X + Y = H_L$. In other words, the constraints are:

$$M \sum_{i=0}^{\tau-1} C_i^M p_s^i (1-p_s)^{M-i} = X \tag{B.6}$$

$$(N-M) \sum_{i=\tau}^{M} C_i^M p_n^i (1-p_n)^{M-i} = Y \tag{B.7}$$

Since both sums are the tails of binomial distributions, the logarithm of the left-hand sides can be replaced to a good approximation by the largest term of each:

$$log(M) + log(C_{\tau-1}^M p_s^{\tau-1}(1-p_s)^{M-\tau+1}) = log(X) \tag{B.8}$$

$$log(N-M) + log(C_\tau^M p_n^\tau (1-p_n)^{M-\tau}) = log(Y) \tag{B.9}$$

If M and τ are large enough, it can be assumed that $\tau - 1 \simeq \tau$ and Stirling's approximation can be used for the combinations:

$$\tau log(\frac{\tau}{Mp_s}) + (M-\tau) log(\frac{M-\tau}{M(1-p_s)}) = log(\frac{M}{X}) \tag{B.10}$$

$$\tau log(\frac{\tau}{Mp_n}) + (M-\tau) log(\frac{M-\tau}{M(1-p_n)}) = log(\frac{N-M}{Y}) \tag{B.11}$$

Then subtracting one from the other:

$$\tau log(\frac{p_s}{p_n}) + (M-\tau) log(\frac{1-p_s}{1-p_n}) = log(\frac{(N-M)X}{MY}) \tag{B.12}$$

This gives an expression for τ in terms of R_s, r:

$$\tau = \frac{log(\frac{(N-M)X}{MY})}{Mlog(\frac{p_s(1-p)}{p(1-p_s)})} \simeq \frac{log(\frac{NX}{MY})}{Mlog(\frac{1-R_s r}{F^2 R_s})} \tag{B.13}$$

From this equation, it can be seen that increasing r (and hence decreasing p) means that τ should decrease as well.

Eqs B.10 and B.11 in principle provide a simpler Langrangian constraint with which to maximise ρ - since Eq B.11 is independent of R_s, it can be used to obtain $\tau = \phi(r)$, which in turn can be substituted into Eq B.10 to obtain a constraint $R_s = \psi(r)$. However, it is still not particularly attractive to tackle.

A further trick would be to equate expressions for the threshold in Eq B.5 and Eq B.13. This gives R_s as a function of p:

$$R_s \simeq F^{\frac{1}{M^2}} \frac{p}{F^2} \tag{B.14}$$

and hence a more tractable constraint.

B.3 Performance

Although the Lagrangian differentiation has not been performed, empirical results show optimal $p = O[0.1]$ (from numerical analysis). Using Eq B.14, the predicted $R_s \simeq 500$. This is over twice as large as simulated spans, but at least of the right order of N.

This illustrates the poorness of simply assuming an $E[H_O] = 2$ comprises exactly one spurious error and one omission error, and of choosing a threshold from distribution intersections, given there are many more chances of spurious errors than omission errors. However, as is common with complex systems like neural networks, this appendix highlights the point where analytical methods must be replaced by numeric ones, as in Section 5.1.2.

Appendix C

User Manual For Willshaw Net Simulator

C.1 Getting Started

The Willshaw Net Simulator (WNS) is initiated by entering "`wns`" from a Unix prompt. The user-interface is in a command-line format, where the user types one-letter commands with a number of arguments, executed upon pressing `Return`. (Alternatively, the WNS can be initiated together with a single Unix argument - the name of a file which contains a sequence of such stored commands to be read in and executed.)

During a session with WNS, two ASCII "output" files will be created, in the directory from which WNS is initiated: `wns.raw` and `wns.pats` - the contents of these will be described later. The user may also ouput further "graph" files via WNS commands, providing the filename as an argument to the command. These files can be read and displayed graphically by Xgraph, which is called automatically from within WNS.

WORD OF WARNING: In a long session, where many thousand patterns are tested on the Willshaw Net (WN), the two Output files may get very large (though they can be "cleared" by a simple command).

C.2 Introducting WNS via an Example Session

After initiating WNS, the User is faced with the `->` prompt as below:

```
Willshaw Net Simulator Vn 2.2
Commands [c.d.f.n.o.q.r.t.!.?]
(Type '?' for help on commands)
->
```

There are 34 basic commands, grouped into 10 basic categories, which are shown upon entering WNS. Each category is selected by a single lower-case letter. The commands within a category are then selected by a second letter - taken together these letters are normally the first letters of the commands they abbreviate - so, for example, the command to "Clear Weights" is **cw**.

The first thing to do is choose various "net parameters". These commands are in the **n** category.

For example, to change the number of input units (N_I) and the number of output units (N_O), and make a N_IxN_O net of 100x200, ie 20,000 switches, the command is **nn**:

```
-> nn 100 200
100 by 200 Willshaw Net
->
```

Note that there must be at least one space between the two integer arguments (though there need not be between the command letters).

When changing the size of the net like so (*e.g.* from the default 64x64), it may be necessary to set a new pattern coding for the N_I- and N_O-dimensional input and output pattern vectors. The pattern coding determined by choosing the number of components of the vectors that are set to 1 rather than 0. This number is usually labelled M_I or M_O for input or output patterns respectively - and the necessary command is **nm**, with first argument M_I and second argument M_O:

```
-> nm 10 20
Pattern coding in=10, out=20. Threshold now 10.
->
```

Changing M_I will also reset the threshold at the same time. (The threshold can be changed independently by a separate command described in the next section.)

Note that if a command is not entered in the correct format, an error will be reported, and the command will have to be retyped. The error reports are not very sophisticated, but some examples are shown after pressing `Return` subsequent to the following invalid inputs:

```
-> nm a 20
!! Invalid Arguments to command ‘nm’

-> nn 100 nm 15 15
!! Invalid Arguments to command ‘nn’

-> nm 10 2000
!! Impossible pattern coding: in=10 and out=2000

-> m 10 20
!! Unknown command character ’m’
!! Unknown command character ’1’
!! Unknown command character ’0’
!! Unknown command character ’2’
!! Unknown command character ’0’
->
```

The next step is to store a number of input/output pattern pairs to be associated by the net. These can be generated randomly, and independently in the (default) case of Hetero-association. For Auto-association, the input and output patterns are identical. Alternatively patterns can be read in from a file.

In the former case, the necessary command is **np**. Its arguments are the first and last pattern pair (inclusive) to be generated - the patterns are thus indexed by a “pattern number” in this range (which, in the ’C’ spirit, starts at 0).

The user may also want to clear the weight matrix with the command **cw** (no arguments) - although this is not strictly necessary at the start of this session. Consequently, the User might enter (noting how more than one command can be executed from a single command line):

```
-> np 0 100 cw
Random patterns 0 to 100 set
Weight Matrix cleared
->
```

Thus 101 input and output patterns have been set; vectors of 100 and 200 components, each with a pattern coding of 0.1. Furthermore, the weight matrix has been cleared (none of these patterns have yet been "learnt").

When setting new blocks of patterns, it is advisable to order them in a contiguous manner. In other words, if pattern numbers 0 to 30 have already been set, and a further block of 20 is desired, the new patterns should be numbered from 31 to 50. Then pattern numbers 0 to 50 inclusive will have been stored. The full range of pattern numbers should always start at 0.

The current WNS settings can be obtained via the command **!**:

```
->!
#Current Willshaw Net Parameters:
#
#Units In = 100                  Units Out = 200
#Coding In = 10                  Coding Out = 20
#
#Patterns Set = 101              and Trained = 0
#Pvalue, Initial = 0.000         Current = 0.000
#
#Normal Training
#
#Hamming Limit = 2
#
#Output to wns.raw and screen - Display Option 1
#
#
#
#Commands [c.d.f.n.o.q.r.t.!.?]
->
```

All of these parameters are explained later.

The net is now ready to be trained and tested on some or all of the patterns set. Training and testing can be performed separately, but there is a useful composite command **rs** ("Run Span"). This command takes 5 arguments. The first two are the first and last pattern numbers to be trained respectively. The third is the size of the "window" for testing patterns - ie the number of patterns to be tested, counting backwards from the last one trained. The fourth argument is the "test-step" - ie the number of patterns to be trained before testing each pattern in the window. The final argument is the Noise factor to be added to each input (cue) pattern during testing. This is explained below.

An example command could be:

```
-> rs 0 100 30 5 0
```

which results in screen output like this:

```
Train=4, Test=0, Ham=0, Noise=0, Pvalue=0.049
Train=4, Test=1, Ham=0, Noise=0, Pvalue=0.049
Train=4, Test=2, Ham=0, Noise=0, Pvalue=0.049

...

--> press Return to continue...

...

Train=99, Test=99, Ham=9, Noise=0, Pvalue=0.636
```

and finishes with:

```
Patterns 0 to 100 trained:
last 30 tested every 5 patterns trained, with 0 Noise.

->
```

The output sent to the screen is paused, and can be resumed each time by pressing `Return`; else pausing can be disabled by typing '`c`'. Similar output, or "raw" data, is also saved to the output file `wns.raw`. *Raw data is only ever produced when a pattern is TESTED on the net.* In the above example, the last 30 patterns are tested every 5 new patterns trained - making a total of 600 lines of raw data.

In each output line, the first field represents the last pattern trained (which is equal to the total number of patterns trained if the training starts from pattern number 0). The second is the pattern number that has just been tested (giving rise to that output line). The third is the Output Hamming Distance (OHD) between the stored (correct) output pattern, and the output of the net, given the (noisy) input pattern.

Option	*Name*	*Description*
1	span	The Number of Patterns Tested, within the window, with an OHD less than the current Hamming Limit (the default is 2), vs. the Number of Patterns Trained.
2	avhd	The Average OHD of Patterns Tested, within the window, vs. the Number of Patterns trained.
3	soc	The Serial Order Curve: the Average OHD of a Pattern, vs. the Serial Order: Number of Subsequent Patterns Trained (up to the size of the window).
4	p	The Loading Density, Pvalue, vs. the Number of Patterns trained.
5	uu	The Unit Usage: the Number of Triggered Switches in an Output Line, vs. the Output Unit Number.

Table C–1: WNS Graph Options

The fourth output field is the noise level. This integer must lie between 0 and N_I. The value is the number of randomly-chosen input components that are flipped to 1 or 0 with equal probability. A value of 0 means a perfect reproduction of the input pattern over the input units; a value of N_I means a net input essentially uncorrelated with the input pattern.

The last field, `Pvalue`, lies between 0.0 and 1.0, and represents the "loading density" of the weight matrix - ie the probability that any particular switch has been turned on.

It is now possible to graph this raw data. There are 5 basic measures, or "graph options", that WNS can calculate. These are described in Table C–1.

Plotting such graphs is done in two stages. First the command **fg** is used to process the raw data and output a new "graph file" containing the relevant calculations. The command takes three arguments: the name of the raw data file, the graph option, and the name of the graph file. The abbreviation `r` can be used to access the default raw data file `wns.raw`. To produce a “span” graph file, the command is:

```
-> fg r 1 span1

Average span is 20.450 with variance 81.647
(with noise 0)
Raw data read from wns.raw and option 1 output to file 'span1'
->
```

This has created a graph file `span1` in the working directory, and also calculated the average and variance of the span.

To display this file graphically, the command is **dg**. This requires the graph option and the graph file name:

```
-> dg 1 span1

Average span is 20.450 with variance 81.647

Data read from span1 and being graphed...
->
```

This should produce an Xgraph window on the screen, showing the “span” of the given WN (which, incidentally, is not so meaningful in this example, since there is no forgetting or unlearning accompanying the training of patterns). The X Window also contains options to save or print a Postscript version of the graph.

It is also useful to copy the default raw data file to a new file, in case the User forgets to clear `wns.raw` at some point, or quits and reenters WNS (in which case `wns.raw` will be reopened and its previous contents will be lost). This is achieved with the command **fr**, giving a suitable filename, eg:

```
-> fr out1
Raw data copied to file ‘out1’
->
```

This raw data can again be processed and plotted by the **fg** and **dg** commands:

```
-> fg out1 3 soc1
Raw data read from ‘out1’ and option 3 output to file ‘soc1’
-> dg 3 soc1
Data read from soc1 and being graphed...
->
```

Note again that such raw data files can be very large, whereas the Graph files are considerably smaller. So, having saved several Graph files from a session, containing the "processed" data, it is often unnecessary to keep the raw data file as well (it can be cleared).

WNS can be exited by entering the command **q**.

This simple example session has covered the basic commands. It shows the sequence of commands in a typical 'run', although it does not show the selection of various palimpsest schemes (for which purpose of comparison the WNS was designed).

A summary of all the possible commands can be displayed via the help command **?**. They are explained in more detail below.

C.3 The WNS commands

Each command is tagged by one or two command letters. A command can have none, one or several arguments. Each argument must be separated by white space, and can be an integer, floating point number, a character or string of alphanumerical characters.

The commands are described below, in the format:

Command Letter(s) *Arguments* Command

Description

Clearing data:

cf (no arguments) Clear Files

Clear the default output files `wns.raw` and `wns.pats`.

cp (no arguments) Clear Patterns

Clear the Pattern Store - ie delete all previously set patterns.

cw (no arguments) Clear Weights

Clear the weight matrix - turning all switches on with probability P(initial) (see command **ni**) and off otherwise.

Displaying Data:

dg *integer1 string1* Display Graph

Plot the graph option *integer1* from the graph file *string1*.

This command calls the Xgraph program to produce a pleasing plot of the relevant data. It is spawned as a separate process, and control continues at the WNS prompt (so several plots can be simultaneously displayed on the screen).

The Xgraph window can be closed by clicking on the `Close` button. A hardcopy can also be obtained by chosing the relevant options offered after clicking on the `Hardcopy` button. Xgraph also has the facility to "expand" an area of the graph, by holding down the mouse button and dragging out an area of the original plot.

dh *integer1 integer2 integer3* Display Hamming

This command prints the hamming distance between two stored input or output patterns in pattern pairs *integer2* and *integer3*. *Integer1* is a flag to index whether input or output patterns of are to be compared - a value of 0 codes for input patterns; a value of 1 codes for output patterns.

A value of -1 for either *integer2* or *integer3* codes for the current input or output of the net, rather than a stored pattern. For example, if pattern pair 17 was the last pair tested, then the command:

```
dh 0 17 -1
```

would compare the stored input pattern number 17 with the input actually used on the net (the hamming distance is only likely to be 0 if non-zero noise was used in the testing).

dp *integer1* Display Pattern

Display Pattern Pair number *integer1*.

This command will print the input and output patterns to the file `wns.pats` and also the screen if the relevant display option is set (see below).

dw (no arguments) Display Weights

Display the current state of the Switch Matrix.

This command will print the weight matrix to the file `wns.pats` and also the screen if the relevant display option is set (see below). Note that for a large net size, the printout may be difficult to interpret, since the matrix rows will be printed over several lines.

File i/o commands:

fc *string1* File Commands

Read commands from file *string1*.

Commands will continue to be read in and executed until the first new-line character is encountered. Control is returned to the WNS prompt (unless an error occurs or the **q** command is read).

Note: all filenames have a maximum of 32 characters.

fg *string1 integer1 string2* File Graph

Process Raw Data and create a Graph File.

This command reads raw data file *string1* and calculates information for graph option *integer1* (see Table C–1). This information, or "processed data", is saved to file *string2*.

The raw data file may either be accessed from an output file previously created via the **fs** command, or the default `wns.raw` file can be used. There is an shorthand for the latter:- *string1* can be abbreviated to `r` in this case.

fp *string1* File Patterns

Read input/output pattern pairs from the file *string1*.

This command is used to enter pre-prepared input and output patterns into the Pattern Store. They will be numbered from 0 onwards - any existing patterns in the store will be over-written.

The format for pattern files is as follows. The size of the patterns and the pattern codings are fixed at the relevant Parameter Settings prior to use of the **fp** command - ie each input pattern must have N_I components, M_I '1's and N_I-M_I '0's, and each output pattern N_O components, M_O '1's and N_O-M_O '0's (these values are checked as each pattern is read in).

Each pattern must consist of a unbroken sequence of '0' or '1' characters, terminated by a new-line character. The pattern pair consists of the input pattern followed by the output pattern, and is separated from the next pattern pair by at least one new-line character.

The format is thus very rigid - the only white space being new-line characters: exactly one separating an input from an output pattern in a pair, and at least one separating the output pattern of one pair from the input pattern of the next.

fr *string1* File Raw
Copy Raw Data to a separate file.

This command will create an new file with name *string1*, which is a copy of the current `wns.raw` data file. This is useful in case `wns.raw` is cleared, over-written or has superfluous data subsequently appended.

Setting Net Parameters:

na (no arguments) Net Auto-association
A toggle between using the net for Hetero-association of patterns and using it for Auto-association of patterns.

Of course, for Auto-association cannot be toggled, if the user has not set $N_I = N_O$ and $M_I = M_O$ with commands **nn** and **nm**.

nh *integer1* Net Hamming
Set the Hamming Limit to be *integer1*.

The hamming distance between two equi-dimensional, binary vectors is the number of components by which they differ. It is used as an index for comparing net output vectors with the desired output patterns.

The hamming limit is an (exclusive) upper limit on this index, used when calculating and plotting the span performance of a net. An OHD of *integer1* or more does not count as adequate retrieval of an output pattern given the input pattern.

The hamming limit must lie between 0 and N_O. The default is 2.

ni *float1* Net Initial-loading

Set the initial probability of a switch being turned on, P(initial) to *float1*. The initial net loading, Pvalue, is of course equal to P(initial).

Clear the weight matrix as well (whenever the weight matrix is cleared, switches are randomly set with this probability.)

For p(initial)=0.0, training procedes from a Tabula Rasa. This is the default setting.

nm *integer1 integer2* Net M-values

Set the pattern coding.

Integer1 is the number of input pattern components set to 1, M_I, *integer2* is the number of output pattern components set to 1, M_O. Obviously, M_I must lie between 0 and N_I, and M_O between 0 and N_O.

nn *integer1 integer2* Net N-values

Set the Size of the net.

Integer1 is the number of input units, N_I, *integer2* is the number of output Units, N_O.

np *integer1 integer2* Net Patterns

Create random input and output patterns pairs, numbered between *integer1* and *integer2*, and store them in the Pattern Store.

The range is inclusive and contiguous.

If some patterns in the range are already set, they will be over-written.

nt *integer1* Net Threshold

Set the threshold for output units. This must lie between 0 and M_I.

Output Options:

oo (no arguments) Output patterns

Display input and output patterns as well as raw data, after each test.

op (no arguments) Output Pausing

A toggle-command for pausing output echoed to the screen.

or (no arguments) Output Raw

Display only raw data after each test.

os (no arguments) Output Screen

A toggle-command for echoing output to the screen.

ow (no arguments) Output Weights

Display weights, input and output patterns and raw data, after each test.

Running a net simulation:

rl *integer1 integer2* Run Last

Test the last *integer1* patterns trained, with noise level *integer2* - see command **rr**.

rr *integer1 integer2 integer3* Run Run

Run (test) input and output pattern numbers in the range *integer1* to *integer2* inclusive, with noise level *integer3*.

The input pattern is imposed on the net's input units. If the Noise Level is greater than zero, the input pattern will be distorted. *Integer2* represents the number of randomly-chosen input components that are flipped to 1 or 0 with equal probability. A value of N_I means, in the average, a net input uncorrelated with the input pattern.

The net output is then compared against the correct output for pattern pair *integer1*. The resulting OHD is compared against the hamming distance limit (see command **nh** above).

The **rr** command always produces raw data output, which is saved to the `wns.raw` file and may be echoed to the screen as well. A typical test result might be:

```
Train=3, Test=3, Ham=1, Noise=5, Pvalue=0.16
```

The first field shows the total number of pattern pairs trained on the net. The second is the last pattern pair tested. The third is the hamming distance, the fourth the noise level in the test and the fifth is the current Loading Density of the net (the "Pvalue" parameter described under command **ni**).

If the relevant Display Option is set (see output commands), copies of the stored pattern pairs and net input/output will also be output, to the file `wns.pats`: as can printouts of the weight matrix.

rs *integer1 integer2 integer3 integer4 integer5* Run Span

The abbreviated train and test command - or "span" command.

Train patterns in the range *integer1* to *integer2* inclusive, testing the last *integer3* patterns (*integer3* is often called the "window") every *integer4* patterns trained, with noise level *integer5*. *Integer4* is often called the "test-step". See commands **rt** and **rr**.

rt *integer1 integer2*

Train input and output pattern numbers in the range *integer1* to *integer2* inclusive.

Setting the Training method (further details of the following palimpsest schemes can be found in the Project Report):

ta *integer1 float1* Training with Ageing weights

Integer1 is the Critical Age, A_o - or mid-point of the probability-age function. This function is the probability of turning off a switch against the age of a switch. It is sigmoidal in shape, rising from a value of 0.0 to a value of 1.0. Entering -1 tags the default value of a critical age based on capacity under normal training, R_c.

Float1 is the Sharpness, D, of the sigmoidal function. Entering -1.0 tags the default infinite sharpness, which is a Heaviside step-function.

tn (no arguments) Training Normal

This sets the standard (non-palimpsest) training scheme.

tr *float1 float2* Training with Random resetting

Float1 is the probability of turning off a switch, r. A value of -1.0 indicates use of the default parameter value based on a loading Pvalue of 0.5 and default z value (below).

Float2 is the probability of turning on a switch, z. A value of -1.0 indicates use of the default value of 1.0.

tu *float1 float2 float3 float4* Training with Unlearning (generalised learning)

Float1 to *float4* are the probabilities w,x,y,z respectively from Section 5.3.1 of the Project Report. Their default values, again set by entering -1.0 for an argument, are given by the Covariance rule.

Other commands:

! (no arguments)

This command will display the current Net Parameter and Output Settings.

? (no arguments)

This command will give a some limited help information - such as a summary of the commands and their arguments. It is no substitute for this User Manual however.

q (no arguments)

This command quits the WNS.

C.4 Technincal Manual

The current WNS is Version 2.2. The code is commented, mainly with implementational details, but assumes knowledge of the terminology adopted in the Project Report.

The files comprising the WNS are:

```
wns.c
wns.h
wui.c
random.c
```

These represent the core, header, user-interface and random number generating files respectively. The header and user-interface files are "included" by `wns.c`. Compilation is thus *via* :

```
gcc wns.c random.c -o wns -lm -O2
```

The WNS uses C libraries:

```
<string.h>
<sys/wait.h>
<stdlib.h>
<stdio.h>
<math.h>
```

and Xwindows software:

```
Xgraph
```

Appendix D

User Manual For Hopfield Net Simulator

The Hopfield Net Simulator is very similar to the Willshaw Net Simulator. Consequently, only the differences will be highlighted here, and the reader is urged to consult the WNS User Manual for full details.

The WNS and HNS have not been tied together under a common user-interface, because there would be no gain in efficiency or productivity. A common user-interface might only become useful when there are more than two models implemented, or if the user had control over the actual model *classes* rather than simply the particular *net* parameters.

The main differences between the WN and HN are:

- The HN has only one layer, so the command **nn** takes only one argument.
- It can only ever function under autoassociation. Thus there is no **na** command, and each Pattern Number tags a single pattern (not an input/output pair).
- Weights in the HN are not binary, but real-valued, and can be negative. Consequently, there is no `Pvalue` variable.
- Updating is not parallel, but an iterative (asynchronous) process.
- All patterns in the HN have components randomly set to '1' with a given probability (and '-1' otherwise). There is no command **nm**. Rather, when

setting the patterns with the command **np**, a third argument must be provided, which is this probability (normally 0.5).

- There is an additional measure of similarity between patterns (vectors): the Overlap index, or Cartesian inner product (see Project Report). This allows two (possibly different) reliable retrieval criteria - in terms of hamming distance or in terms of overlap. These citeria are by the command **nr**.
- HN units have a fixed threshold of 0.
- The variable "eta" in the HN Learning Rule can be set by the command **ne**.

D.1 Getting Started

The Hopfield Net Simulator (HNS) is initiated by entering "`hns`" from a Unix prompt. The user-interface is in a command-line format, where the user types one-letter commands with a number of arguments, executed upon pressing `Return`. (Alternatively, the HNS can be initiated together with a single Unix argument - the name of a file which contains a sequence of stored commands to be read in and executed.)

During a session with HNS, two ASCII "output" files will be created, in the directory from which HNS is initiated: `hns.raw` and `hns.pats` - the contents of these will be described later. The user may also ouput further "graph" files via HNS commands, providing the filename as an argument to the command. These files can be read and displayed graphically by Xgraph, which is called automatically from within HNS.

WORD OF WARNING: In a long session, where many thousand patterns are tested on the Hopfield Net (HN), the two Output files may get very large (though they can be "cleared" by a simple command).

D.2 Introducting HNS via an Example Session

See the relevant section in the Willshaw Net User Manual.

There are 36 basic commands, grouped into exactly the same 10 basic categories as in the WNS.

The default HNS settings can be obtained via the command **!**:

```
Hopfield Net Simulator Vn 2.2
Commands [c.d.f.n.o.q.r.t.!.?]
(Type '?' for help on commands)
-> !
#Current Hopfield Net Parameters:
#
#Units = 100                   Pattern Coding = 0.500
#
#Initial Weight Range = -0.000000 to 0.000000
#
#Patterns Set = 0              and Trained = 0
#Normal Training, Eta = 1.000000
#
#Overlap Limit = 0.970         Hamming Limit = 2
#
#Output to hns.raw and screen - Display Option 1
#
#
#
#Commands [c.d.f.n.o.q.r.t.!.?]
->
```

The raw data contains the additional overlap field:

```
...
Train=99, Test=97, Hamming=29, Overlap=0.4200, Noise=0
Train=99, Test=98, Hamming=33, Overlap=0.3400, Noise=0
Train=99, Test=99, Hamming=22, Overlap=0.5600, Noise=0
...
```

There are 6 basic measures, or "graph options", in the HNS, as compared to the 5 in the WNS. They are described in Table D–1. Note that the calculation of Pvalue is not possible, Unit Usage has been replaced by Incident Weights, and two additional plots for Average Overlap and Serial Order Curve with Overlap (rather than OHD) are provided.

Option	*Name*	*Description*
1	span	The Number of Patterns tested, within the window, that can be reliably retrieved (depending on OHD or overlap criterion set), vs. The Number of Patterns trained.
2	avhd	The Average OHD of Patterns Tested, within the window, vs. The Number of Patterns trained.
3	avol	The Average Output Overlap of Patterns Tested, within the window, vs. The Number of Patterns trained.
4	sochd	A Serial Order Curve: the average OHD of a Pattern, vs. the Serial Order: number of subsequent patterns trained (up to the size of the window).
5	socol	A Serial Order Curve: the average Output Overlap of a Pattern, vs. the Serial Order.
6	wgts	The Incident Weights: the average weight value incident on a Unit, vs. The Unit Number.

Table D–1: HNS Graph Options

D.3 The HNS commands

Each command is tagged by one or two command letters. A command can have none, one or several arguments. Each argument must be separated by white space, and can be an integer, floating point number, a character or string of alphanumerical characters.

The commands are described below, in the format:

Command Letter(s) *Arguments* Command
Description

Clearing data:

cf (no arguments) Clear Files
Clear the default output files `hns.raw` and `hns.pats`.

cp (no arguments) Clear Patterns
Clear the Pattern Store - ie delete all previously set patterns.

cw (no arguments) Clear Weights
Clear the weight matrix.

Displaying Data:

dg *integer1 string1* Display Graph
Plot the graph option *integer1* from the graph file *string1*.

This command calls the Xgraph program to produce a pleasing plot of the relevant data. It is spawned as a separate process, and control continues at the HNS prompt (so several plots can be simultaneously displayed on the screen).

The Xgraph window can be closed by clicking on the `Close` button. A hardcopy can also be obtained by chosing the relevant options offered after clicking on the `Hardcopy` button. Xgraph also has the facility to "expand" an area of the graph, by holding down the mouse button and dragging out an area of the original plot.

dh *integer1 integer2* Display Hamming
This command prints the hamming distance between two stored patterns *integer1* and *integer2*.

A value of -1 for either *integer1* or *integer2* codes for the current state of the net, rather than a stored pattern. For example, if pattern 17 was the last pair tested, then the command:

```
dh 17 -1
```

would compare the stored pattern number 17 with the net output.

do *integer1 integer2* Display Overlap
This command prints the overlap between two stored patterns *integer1* and *integer2*.

dp *integer1* Display Pattern
Display Pattern Pair number *integer1*.

This command will print the pattern *integer1* to the file `hns.pats` and also the screen if the relevant display option is set (see below).

dw (no arguments) Display Weights
Display the current state of the Weight Matrix.

This command will print the weight matrix to the file `hns.pats` and also the screen if the relevant display option is set (see below). Note that for a large net size, the printout may be difficult to interpret, since the matrix rows will be printed over several lines.

File i/o commands:

fc *string1* File Commands
Read commands from file *string1*.

Commands will continue to be read in and executed until the first new-line character is encountered. Control is returned to the HNS prompt (unless an error occurs or the **q** command is read).

Note: all filenames have a maximum of 32 characters.

fg *string1 integer1 string2* File Graph

Process Raw Data and create a Graph File.

This command reads raw data file *string1* and calculates information for graph option *integer1* (see Table D–1). This information, or "processed data", is saved to file *string2*.

The raw data file may either be accessed from an output file previously created via the **fs** command, or the default `hns.raw` file can be used. There is an shorthand for the latter:- *string1* can be abbreviated to `r` in this case.

fp *string1* File Patterns

Read patterns from the file *string1*.

This command is used to enter pre-prepared patterns into the Pattern Store. They will be numbered from 0 onwards - any existing patterns in the store will be over-written.

The size of the patterns is fixed by the parameter setting from the last **nn** command prior to use of the **fp** command - ie each pattern must have N components of either '1' or '0' ('0' is the program's encoding of the spin value '-1').

Each pattern is terminated by at least one new-line character. The format is thus very rigid - the only white space being new-line characters.

fr *string1* File Raw

Copy Raw Data to a separate file.

This command will create an new file with name *string1*, which is a copy of the current `hns.raw` data file. This is useful in case `hns.raw` is cleared, over-written or has superfluous data subsequently appended.

Setting Net Parameters:

ne *float1* Net Eta-value

Set the value of "eta" in the Learning Rule. Its value must lie between 0.0 and N.

nh *integer1* Net Hamming

Set the Hamming Limit to be *integer1*. The hamming limit must lie between 0 and N.

nh *float1* Net Overlap

Set the Overlap Limit to be *float1*. The overlap limit must lie between 0.0 and 1.0.

ni *float1* Net Initial weight range

Set the initial weights to lie randomly in the range −*float1* to +*float1*.

Clear the weight matrix as well (whenever the weight matrix is cleared, switches are randomly set with this probability.)

For *float1* = 0.0, training procedes from a Tabula Rasa. This is the default setting.

nn *integer1* Net N-value

Set the Size of the net.

Integer1 is the number of units in the net, N.

np *integer1 integer2 float1* Net Patterns

Create random patterns, numbered between *integer1* and *integer2*, and store them in the Pattern Store, each with expected pattern coding *float1*.

The range is inclusive and contiguous.

If some patterns in the range are already set, they will be over-written.

nr (no arguments) Net Retrieval criterion

Toggle between using the Hamming or Overlap Limits for span determination.

The default is to use the overlap metric.

Output Options:

oo (no arguments) Output patterns

Display patterns, as well as raw data, after each test.

op (no arguments) Output Pausing

A toggle-command for pausing output echoed to the screen.

or (no arguments) Output Raw

Display only raw data after each test.

os (no arguments) Output Screen

A toggle-command for echoing output to the screen.

ow (no arguments) Output Weights

Display weights, patterns and raw data, after each test.

Running a net simulation:

rl *integer1 integer2* Run Last

Test the last *integer1* patterns trained, with noise level *integer2* - see command **rr**.

rr *integer1 integer2 integer3* Run Run

Run (test) input and output pattern numbers in the range *integer1* to *integer2* inclusive, with noise level *integer3*.

The cue pattern is imposed on the net's units. If the noise level is greater than zero, the pattern will be distorted. *Integer2* represents the number of randomly-chosen components that are flipped to 1 or 0 with equal probability. A value of N means, in the average, a net input uncorrelated with the presented pattern.

The net output, or final stable state, is then compared against the pattern *integer1*, through the hamming or overlap limit.

The **rr** command always produces raw data output, which is saved to the `hns.raw` file and may be echoed to the screen as well. A typical test result might be:

```
Train=99, Test=99, Hamming=22, Overlap=0.5600, Noise=0
```

The first field shows the total number of patterns trained on the net. The second is the last pattern tested. The third is the hamming distance, the fourth the overlap and the fifth is the Noise Level.

If the relevant Display Option is set (see output commands), copies of the stored pattern and net output will also be dumped to the file `hns.pats`: as can printouts of the weight matrix.

rs *integer1 integer2 integer3 integer4 integer5* Run Span
The abbreviated train and test command - or "span" command.

Train patterns in the range *integer1* to *integer2* inclusive, testing the last *integer3* patterns (*integer3* is often called the "window") every *integer4* patterns trained, with noise level *integer5*. *Integer4* is often called the "test-step". See commands **rt** and **rr**.

rt *integer1 integer2*
Train pattern numbers in the range *integer1* to *integer2* inclusive.

Setting the Training method (further details of the following palimpsest schemes can be found in the Project Report):

ta *float1* Training with Attenuated weights
Float1 is the value for "lambda", or attenuation factor. Entering -1.0 tags the default, optimal attenuation for current net size.

tb *float1* Training with Bounded weights
Float1 is the value for "B", or weight bound. Entering -1.0 tags the default, optimal bound for current net size.

te (no arguments) Training with Enforced storage
This sets the Enforced Storage palimpsest scheme.

tn (no arguments) Training Normal
This sets the standard (non-palimpsest) training scheme.

tr *integer1 integer2 float1* Training with Random unlearning

Integer1 is the number of learning episodes per unlearning episode (default is 1). *Integer2* is the number of unlearning trials performed per unlearning episode (default is 10). *Float1* is the weight decrement for each unlearning trial (default is 0.1).

Other commands:

! (no arguments)

This command will display the current Net Parameter and Output Settings.

? (no arguments)

This command will give a some limited help information - such as a summary of the commands and their arguments. It is no substitute for this User Manual however.

q (no arguments)

This command quits the HNS.

D.4 Technical Manual

The current HNS is Version 2.2. The code is commented, mainly with implementational details, but assumes knowledge of the terminology adopted in the Project Report.

The files comprising the HNS are:

```
hns.c
hns.h
hui.c
random.c
```

These represent the core, header, user-interface and random number generating files respectively. The header and user-interface files are "included" by `hns.c`. Compilation is thus *via* :

```
gcc hns.c random.c -o hns -lm -O2
```

The HNS uses C libraries:

```
<string.h>
<sys/wait.h>
<stdlib.h>
<stdio.h>
<math.h>
```

and Xwindows software:

```
Xgraph
```

Appendix E

Simulation runs in this project

The results quoted in this project need some qualification of the simulation parameters used to generate them.

"Runs" are conducted by initiating the Simulators with a pre-prepared command file, usually in the "background" and with the Unix `nohup` command. This is because simulations with $N = 512$ and $P = 10,000$ training patterns, usually take of the order of 5 hours on a Sun-4.

E.1 WNS

A typical file for studying two cases of Random Resetting might be:

```
op os nn 512 512 nm 9 9 np 0 10000 ni 0.5 tr -1 -1 rt 0 499 rs 500
10000 500 100 0 fg r 1 512span_rr50 fg r 3 512soc_rr50 fg r 4
512p_rr50 cw cf ni 0.44 tr 0.000395 1 rs 0 10000 500 100 0 fg r 1
512span_rr44 fg r 3 512soc_rr44 fg r 4 512p_rr44 cf q
```

Note that there are no new line characters in the actual input files. Note also that this creates (temporarily) a `wns.raw` file with 50,000 lines of data, each line with at least 15 characters, so considerable file-space is needed.

Several points are worth noting:

- The window size of 500 is ten times bigger than the expected span. (See Fig 5–1 in the Project Report).

- The span will be averaged over 95 measurements.

- Weights and (raw) files are cleared between "runs".

- Screen output has been turned off and pausing removed (output that is always sent to screen will be sent to the file `nohup.out` instead, like average span data for example).

- Graph files are actually plotted after the run has finished and `nohup` has terminated, through interactive use of WNS and the **dg** command.

- The asymptotic switch distribution with $Pvalue = 0.5$ is set with the command **ni**, since the estimate of span does not want to be biased by any transient effects (as when training from a Tabula Rasa for example).

- The first 500 patterns are "pre-trained" without testing, so that the testing of the first window, when $R = 501$, is not over a random background of switches. (Note to pre-train from $Pvalue = 0.0$ to $Pvalue = 0.5$, rather than using **ni**, is also possible, but would take about five times as many patterns.)

Note that the use of **ni** is not appropriate for the Ageing Weights scheme, since each switch needs to acquire an associated age. Here, a relevant file here would be:

```
op os nn 512 512 nm 9 9 np 0 10000 ta 1900 -1 rt 0 1999 rs 2000
10000 2000 100 0 fg r 1 512span_aw50 fg r 3 512soc_aw50 fg r 4
512p_aw50 cf q
```

Then roughly the first $0.44 \times 512 \times 512$ switches will have ages between 1 and 1900, so the first testing will be over 2000 pre-trained patterns.

E.2 HNS

A typical file for studying noisy cues ($n = 10$) in the optimal cases of Attenuated and Bounded Weights might be:

```
op os nn 512 np 0 4000 0.5 nh 26 nr ne 0.00803 ta 0.984 rt 0 1000
rs 1001 4000 100 20 10 fg r 1 512span_aw_10 fg r 4 512soch_aw_10
fg r 5 512soco_aw_10 cw cf ne 0.00586 tb 0.0442 rt 0 1000 rs 1001
4000 100 20 10 fg r 1 512span_bw_10 fg r 4 512soch_bw_10 fg r 5
512soco_bw_10 cf q
```

Several points are worth noting:

- The window size does not need to be so big for HN palimpsest schemes compared to WN schemes (an empirical finding - survival times have less spread).
- The span will be averaged over 150 measurements.
- Weights and (raw) files are cleared between "runs".
- The two serial order curves are simply to provide two perspectives: with hamming distance and with overlap.
- Pretraining (of a somewhat arbitrary $1,001$ patterns) is again employed to remove transients effects of Tabula Rasa training.

E.3 Randomness

Finally, it may be noted that the user has no control over the random generation of patterns (after N and M have been selected), *e.g.* the user cannot change the random seeds without explicitly altering the file `random.c`. This is because changing the seeds without knowledge of the restrictions can produce undesireable results.

However, there will typically be many calls to the random number generator (*e.g.* $2N$ times for each pattern pair, N times for every noisy cue, up to N^2 times for each Random Resetting episode, *etc.*). This means there will naturally be considerable variability between different runs, without necessarily having to change the seeds, *i.e.* results can only be replicated exactly if an identical sequence of commands is used.

The file `random.c` was created by R.A.O'Keefe from an algorithm AS 183 from the journal "Applied Statistics". It has been briefly tested with the seeds provided. For example, testing the hamming distance between 10000 patterns when $N = 512$ and $M = 256$ gives the expected normal distribution, as shown in Fig E–1. Note the generation of random patterns with exactly M of N components set is achieved through Floyd's algorithm.

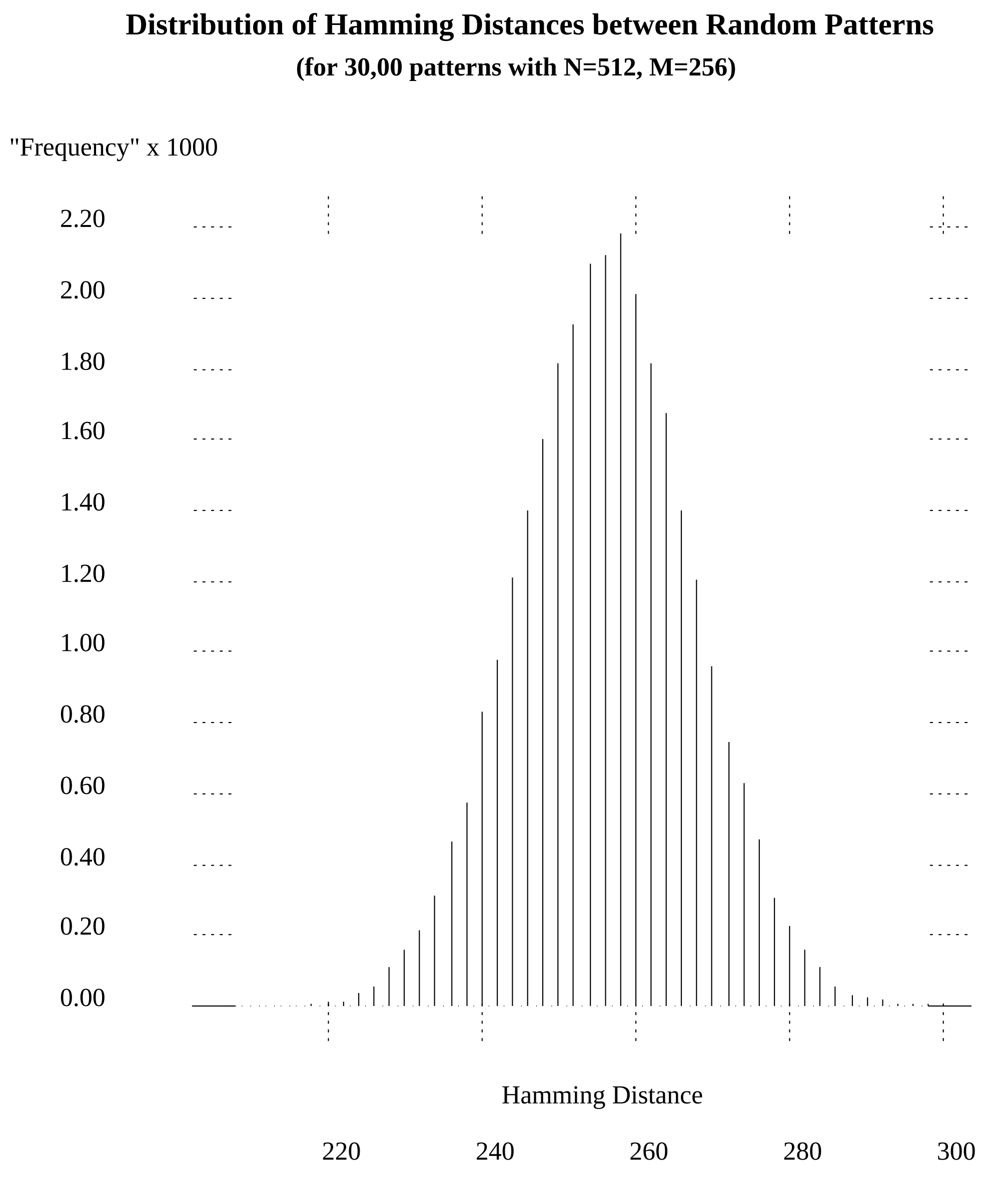


Figure E–1: Distribution of Hamming Distances between Randomly Generated Patterns