ELSEVIER

Information Sciences 111 (1998) 65–81

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

Experimental study of Perceptron-type local learning rule for Hopfield associative memory

Arun Jagota *, Jacek Mańdziuk [1]

International Computer Science Institute, University of California, Berkeley, CA 94720, USA

Received 17 July 1997; received in revised form 23 October 1997; accepted 30 December 1997

Abstract

Kamp and Hasler proposed in 1990 a Perceptron-type learning rule for storing binary patterns in the Hopfield associative memory. They also proved its convergence. In this paper this rule is evaluated experimentally. Its performance is compared with that of the commonly-used Hebb rule. Both rules are tested on a variety of randomly generated collections of library patterns parametrized by the number of patterns M in a collection, the density p of the patterns, and the measure of correlation B of the bits in a pattern. The results are evaluated on two criteria: stability of the library patterns, and error-correction ability during the recall phase. The Perceptron-type rule was found to be perfect in ensuring stability of the stored library patterns under all evaluated conditions. The Hebb rule on the other hand was found to degrade rapidly as M was increased, or the density p was decreased, or the correlation B was increased. For not too large M, the Perceptron-type rule was also found to work much better, under a variety of conditions, than the Hebb rule in correcting errors in probe vectors. The conditions, when the Perceptron-type rule worked better, included a range of pattern-densities p, a range of correlations B, and several degrees of error e in a probe vector. The uniformly random case ($p = 0.5, B = 1$) was the main exception when the Hebb rule systematically equalled or outperformed the Perceptron-type rule in the error-correction experiments. The experiments revealed interesting aspects of the evolution of the Per-

* Corresponding author. Address for correspondence: Department of Computer Science, 225 Applied Sciences Building, University of California, Santa Cruz, CA 95064, USA. E-mail: jagota@cse.ucsc.edu

[1] Senior Fulbright Scholar visiting ICSI, Berkeley and EECS Dept. University of California, Berkeley under grant no. 20985. Permanent affiliation: Institute of Mathematics, Warsaw University of Technology, Plac Politechniki 1, 00-661 Warsaw, Poland. E-mail: mandziuk@ alpha.im.pw.edu.pl

PII: S0020-0255(98)00005-X

ceptron-type rule. Unlike the Hebb rule, the Perceptron-type rule evolved over multiple epochs (multiple presentations of a library pattern). The error was found not to decrease monotonically, though eventually it always became zero. When M was sufficiently large, it was found that, for the uniformly random patterns, the Perceptron-type rule produced a degenerate weight matrix, whose diagonal terms dominated (stability was achieved as a byproduct).

Keywords: Neural networks; Storage capacity; Error correction; Recurrent networks

1. Introduction

The binary Hopfield network is a well-known model for associative memories [1]. Since its introduction it has attracted a large amount of interest [2] and, despite some of its drawbacks, the network remains an active topic of research. One of the main drawbacks of the original model is its limited capacity under the Hebb rule, which is poor on random patterns [1,3] and even poorer on correlated patterns [2]. Over the past decade, several attempts have been made to improve its capacity [2,4]. One notable example is the pseudoinverse rule, which works well for linearly independent patterns [2]. Unlike the Hebb rule however, the pseudoinverse rule has no biological interpretation, is non-local, and is computationally intensive.

In this paper we experimentally evaluate a simple, online, local Perceptron-type learning rule for the binary Hopfield model. This rule was first presented in [5], together with a proof of convergence. In [5], the authors emphasized the similarities between this rule and the Hebb rule – in particular that if one reformulates the associative memory problem as one involving Perceptron learning, one is led to a rule that is essentially Hebbian. In this paper we emphasize the *differences*. In particular, we view the Perceptron-type rule as a supervised extension of the unsupervised Hebbian rule that incorporates a Perceptron-type term for correcting unstable bits. As an aside, it is interesting to note that there does not appear to have been much work on using Perceptron learning for feedback associative memories. The one work we are aware of in addition to [5,4] is by Liu and Lu [6], who use Perceptron Learning to store memories in a continuous-time feedback network with saturated-linear neurons [6].

The rule retains the simplicity, online nature, and locality (hence computational efficiency) of the Hebb rule. Extensive computer simulations reveal that it performs better in most situations of interest. Unlike the Hebb rule, the Perceptron-type rule works over multiple epochs and often reduces error non-monotonically, over the epochs.

Our main results may be summarized as follows. In extensive computer simulations it was found that in all cases the Perceptron-type rule was able to store (all bits of) all vectors stably. However, the simulations also revealed that, for

uniformly random patterns, when the number of patterns approached $n\sqrt{n}$, n being the pattern dimension (also the number of neurons), the rule produced a diagonally-dominant matrix for which all the 2^n possible vectors were stable. Experiments were conducted to estimate the error-correction ability around the stored library vectors [2] and thereby get a more realistic assessment of the capacity.

The main experimental results are as follows. While the Perceptron-type rule ensured stability of all the library vectors in all tested situations, the Hebb rule degraded rapidly on this measure when the vectors were made sparse, or more correlated in a certain way. When the number of library vectors was kept between $0.1n$ and $0.3n$, the Perceptron-type rule significantly outperformed the Hebb rule in correcting errors in probe vectors derived by flipping one (two) bit(s) at random from the library vectors. In particular, the performance of the Perceptron-type rule remained roughly constant over various degrees of sparseness and correlation of the library vectors, whereas the performance of the Hebb rule, like in the stability case, rapidly degraded when the sparseness or the correlation was increased.

The Perceptron-type rule was found to work significantly better than the Hebb rule even when the probe vectors had large amounts of error. For the case $M = 0.1n$, situations when the probe vectors had 2.5–50% of the bits in error were evaluated. In all cases when the patterns were sparse ($p \leqslant 0.3$) and/or correlated, the Perceptron-type rule significantly outperformed the Hebb rule in correcting errors.

This paper is organized as follows. Section 2 reviews the Hopfield network, then presents the Hebb rule, and finally the Perceptron-type rule. Section 3 presents the experimental studies: Section 3.1 presents the storage capacity results; Section 3.2 the error-correction results. Section 4 discusses potential applications. Section 5 presents the conclusions.

2. Hopfield network, Hebb rule, and Perceptron-type rule

The Hopfield model is a recurrent neural network of n binary neurons. Each neuron has an external state -1 (denoting not firing) or $+1$ (denoting firing). Pairs of neurons are interconnected together via a symmetric weight matrix (i.e. $w_{ij} = w_{ji}$) with non-negative diagonal elements. The Hopfield network computes as follows. All neurons are initially set to some state-vector in $\{-1, 1\}^n$. The network then updates its neuron's states based on some evolution equation. (The network is recurrent because every neuron's state can affect

[2] The vectors that one attempts to store in an associative memory model. Also known as fundamental memories, prototype vectors, memory patterns, etc.

every other neuron's state.) In 1982 Hopfield exhibited a particular evolution equation according to which if the network is operated, convergence is assured to some steady state from an arbitrary initial state. The steady states could, furthermore, be controlled by the choice of the weight matrix $\mathbf{W}$. Hopfield then suggested that this network may be used for storing memories, by appropriate choice of the weight matrix. Given a set $\mathbf{X}^1, \ldots, \mathbf{X}^p$ of vectors of length n one would attempt to construct a suitable weight matrix $\mathbf{W}$ such that the vectors were recorded as steady states of the network. Hopfield proposed the Hebb rule for this purpose. For more details, the reader is referred to [2].

Both the Hebb and the Perceptron-type rule are designed to store a set $\mathbf{X}^1, \ldots, \mathbf{X}^p$ of patterns, where $\mathbf{X}^\mu \in \{-1, 1\}^n$, $\mu = 1, \ldots, p$, in a Hopfield network of n neurons. Both rules are *online*, i.e. the patterns are presented to them one by one. Both rules attempt to store a presented pattern by adjusting the weights of the network (the neuron thresholds remain 0). Both rules are *local* in that changes to weight w_{ij} are based only on activities of neurons i and j. Both rules assume that the initial weight matrix is $\mathbf{W} = 0$.

First consider the Hebb rule. To store the pattern-set $\mathbf{X}^1, \ldots, \mathbf{X}^p$ each pattern $\mathbf{X}^\mu$ is presented exactly once and the weight matrix adjusted as follows:

$$w_{ij}(t+1) := w_{ij}(t) + \eta X_i^\mu X_j^\mu. \tag{1}$$

Unfortunately, though its simplicity is attractive, the Hebb rule's storage capacity is poor [2]. The Perceptron-type rule is a good alternative to explore because it retains several of the attractive features of the Hebb rule – simplicity, locality, online nature, biological relevance – while being expected to perform better because it uses supervised learning to reduce errors. Whether the Perceptron-type rule does in fact perform better or not is the empirical question studied in this paper.

The Perceptron-type rule may be described as the following modification of Eq. (1):

$$w_{ij}(t+1) := w_{ji}(t+1) := w_{ij}(t) + \frac{\eta}{2}[(X_i^\mu - Y_i^\mu)X_j^\mu + (X_j^\mu - Y_j^\mu)X_i^\mu], \tag{2}$$

where $\mathbf{Y}^\mu := \mathrm{sgn}(\mathbf{W}\mathbf{X}^\mu)$.

The Perceptron-type rule may be understood as follows. First imagine that the weights of the network are not required to be symmetric. In this case, the problem of storing the pattern-set $\mathbf{X}^1, \ldots, \mathbf{X}^p$ stably may be formulated as one to find a weight matrix $\mathbf{W}$ satisfying

$$\mathbf{X}^\mu = \mathrm{sgn}(\mathbf{W}\mathbf{X}^\mu) \quad \text{for } \mu = 1, \ldots, p.$$

Because $\mathbf{W}$ is not required to be symmetric, this may be reinterpreted as learning the auto-associative mapping in a Perceptron with n input neurons and n output neurons. Presenting the patterns one by one and applying the online Perceptron algorithm [2] yields the rule

$$w_{ij}(t+1) := w_{ij}(t) + \eta(X_i^\mu - Y_i^\mu)X_j^\mu. \tag{3}$$

The problem of course is that this does not ensure the symmetry of **W** since changes to $w_{ij}(t+1)$ and $w_{ji}(t+1)$ may be unequal. One way to ensure symmetry is to take the average $1/2(\Delta w_{ij}(t+1) + \Delta w_{ji}(t+1))$ of both changes and make this change to both weights. This yields Eq. (2). The fact that rule (2) is convergent [5] is thus not entirely surprising, given its roots in the Perceptron algorithm.

3. Experimental studies

To assess the performance of the Perceptron-type learning rule on associative memories we focus mainly on two questions in the following paper:

- What is the storage capacity of the Perceptron-type rule in relation to that of the Hebb rule?
- What is the error-correction ability of the Perceptron-type rule in relation to that of the Hebb rule?

These questions are addressed in Sections 3.1 and 3.2. First, the library patterns used in the experiments are described.

In order to evaluate storage capacity and error-correction ability under a variety of conditions, a single library pattern-set generator capable of generating a variety of pseudorandom -1/1 patterns was designed, with the following parameters: N the dimension of the patterns, M the number of patterns, and B the block size. The N bits of a pattern were divided into fixed blocks of size B each. Bits 1–B belong to block one, $B+1$–$2B$ to block two, and so on. All bits in a block were given the same value (all 1 for a 1-block, or all -1 for a (-1)-block). p is the probability that an arbitrary block of an arbitrary generated pattern was a 1-block.

The generator produces a pseudo-random $\langle N, p, B\rangle$ pattern as follows. Each of the N/B blocks is independently generated as a 1-block with probability p or as a (-1)-block with probability $1-p$. Varying p controls the sparseness of the patterns, and varying B the correlation (the uncorrelated case being $B=1$).

In all of the experiments reported in this paper, η was set to 1 for both the Hebb rule and for the Perceptron-type rule. In all experiments except the one in Table 2, the pattern-dimension N was set equal to 200.

Table 1 reports the storage capacity results, on pattern-sets for various $\langle M, N, p, B\rangle$ stored via the Hebb rule and via the Perceptron-type rule. Tables 3 and 4 report the error-correction results of the Hebb and the Perceptron-type rule, on the same pattern-sets as those of Table 1.

3.1. The storage capacity results

A *bit* i of a library pattern **X** is called *unstable* if $X_i \neq \mathrm{sgn}(\sum_j w_{ij}X_j)$. A *library pattern* is called *unstable* if even one of its bits is unstable. One may evaluate

Table 1
Storage capacity results

M	B	$p = 0.5$		$p = 0.3$		$p = 0.1$	
		#u	#e	#u	#e	#u	#e
20	1	0.04	3.2	29.46	3.4	20.25	5.6
20	2	0.70	3.3	27.22	3.1	20.44	5.6
20	3	1.36	3.2	21.75	3.4	20.73	5.9
20	4	2.52	3.2	21.32	3.3	20.34	5.5
20	5	3.62	3.4	19.00	3.8	20.32	5.5
40	1	1.24	3.5	50.72	4.6	20.41	7.1
40	2	3.38	4.3	45.21	4.7	20.25	6.7
40	3	4.30	4.6	40.53	4.9	20.49	6.5
40	4	4.48	4.6	35.64	5.0	20.44	5.5
40	5	5.18	4.5	32.61	4.6	20.55	5.5
60	1	3.37	4.8	57.30	5.4	20.06	8.4
60	2	5.40	5.3	52.94	5.8	20.43	7.4
60	3	5.37	6.0	48.54	6.2	20.22	7.9
60	4	5.08	5.8	42.86	5.9	20.41	5.6
60	5	4.82	5.5	38.00	5.0	20.52	5.9

Columns labeled #u denote the average number of unstable bits in a pattern after the pattern-set is stored via the Hebb rule. The quantity #u is averaged over all library patterns (M) and trials (10). Column labeled #e denotes the number of epochs of the Perceptron-type rule to converge to zero error (all library patterns stored stably). The number of epochs is also the number of presentations of each pattern.

storage rules for the Hopfield network based on the percentage of the library patterns that are made unstable. However this does not reveal whether these unstable library patterns were "almost stable" (i.e. only a very few of their bits were unstable) or whether they were very unstable. This distinction is important in some applications, for example image restoration, where one may be willing to tolerate a few unstable bits (i.e., pixels that differ from the perfect image). In this paper we adopted the average number of unstable bits in a library pattern as our measure of storage capacity.

Columns labeled #u in Table 1 report the number of bits of a library pattern unstable after storage of the entire pattern-set via the Hebb rule, averaged over all the patterns in the pattern-set (M), and over all trials (10). This result is not included in Table 1 for the Perceptron-type rule because in all cases it was zero (i.e. in all experiments, all bits of all patterns were stable after storage).

3.1.1. Hebb rule results and analysis

The results of Table 1 reported under the columns labeled #u permit the following observations. The first obvious one is that as M is increased with

all other parameters kept fixed, the number of unstable bits in the library patterns increases significantly. Next, for fixed M and N, the number of unstable bits in the library patterns is very much higher in the sparse cases ($p = 0.1, 0.3$) than in the non-sparse case ($p = 0.5$). Thus the Hebb rule degrades on sparse patterns. Finally, for fixed M, N, and $p = 0.5$ (i.e. the non-sparse case), the number of unstable bits in the library patterns increases monotonically as block size (hence correlation) is increased from 1 to 5. Thus, for $p = 0.5$, the Hebb rule also degrades somewhat as block size is increased. (For $p = 0.3$, this trend is reversed, however this event is not so significant because the number of unstable bits is so high in all cases.)

In addition to the main effects discussed in the previous paragraph, the following secondary effects are also clearly seen in Table 1. In going from $p = 0.3$ to $p = 0.1$ the number of unstable bits in the library patterns in fact decreases somewhat. Thus we see a non-monotone effect. In going from $p = 0.5$ to $p = 0.3$, the Hebb rule degrades significantly. However in making the patterns even more sparse, the performance in fact improves a little. Finally, as p is decreased, the number of unstable bits begins to become relatively invariant to block size changes.

The primary and subtle effect on the performance of the Hebb rule as a function of sparseness (p) is easily explained as follows. When $p = 0.1$, there are very few attractors, perhaps just two: $(-1)^N$, the vector with all components -1, and its complement vector 1^N. Therefore, for every library pattern about $0.1N$ of its bits (the 1 bits) should be unstable. Table 1 reveals that this is indeed the case. Similar reasoning indicates that for $p = 0.3$ we would expect about $0.3N$ bits of a library pattern to be unstable on average. For larger M we see that this is indeed the case from Table 1. This reasoning does not extend to $p = 0.5$ however. The reason is that as p is increased new attractors emerge. Unless M is too large, these attractors are closer to the library patterns. For example when $p = 0.5$ and $M < 0.13N$, 1% or less of the bits in the library patterns are unstable [2].

The effect that as sparseness is increased (p is decreased) the Hebb rule becomes relatively insensitive to block size changes admits the following partial explanation. For fixed p and B define b_i as the random variable whose value is 1 if the ith block of a pseudorandom $\langle p, B\rangle$ N-bit pattern is a 1-block, and 0 otherwise. Define $R = B \times \sum_i b_i$ as the random variable for the number of 1-bits in such a pseudorandom pattern. The expectation of R is

$$B \times \sum_i E[b_i] = B \times pN/B = pN,$$

which is independent of B. Using independence of the b_i, the variance of R is

$$B^2 \times \sum_i \mathrm{Var}(b_i) = B^2 \times p(1-p)N/B = B \times p(1-p)N,$$

which does depend on B. Assume that N is much larger than B and $1/p$. Therefore when p is very small, the expectation of R is a small fraction of N. The variance of R, over the range of $B \ll N$, is also a small fraction of N. Finally, when p is very small, $(-1)^N$ and its complement are expected to be the only attractors in the network after storage of the pattern set. Therefore, as noted earlier, the 1-bits in a library pattern are exactly those that are unstable. In view of all of this, it is reasonable to conclude that when p is very small, the actual number of observed unstable bits in a library pattern varies little with B, provided $B \gg N$, in the sense that the range of number of observed unstable bits is narrow. When $p = 0.5$ on the other hand, the variance of R is more sensitive to the value of B in the sense that the range of variances of R as a function of B is much wider. Furthermore, there are probably many more attractors in these cases, and the number of them and their nature is perhaps also quite sensitive to the value of B.

To this point, we have focused on the case $p = 0.5$ and the sparse cases $p = 0.3, 0.1$. Since the Hebb rule is symmetric and the Perceptron-type rule is almost symmetric [3] we would expect the results to be very similar for the Hebb rule and similar for the Perceptron-type rule in the complementary case, i.e. when $p = 0.7, 0.9$. We do not report the results for these (very-dense) cases here. Table 1 of the conference version of this paper contains them [7]. In that paper, p actually denotes the probability of a -1 value, despite the fact that it is (erroneously) noted in that paper that p denotes the probability of a 1 value.

3.1.2. A closer look at the Perceptron-type rule

A closer examination of the learning process during the experiments revealed three insights. The first insight was that for uniformly distributed random patterns ($\langle p, B \rangle = \langle 0.5, 1 \rangle$), as the ratio M/N was increased, the symmetric matrix became diagonally-dominant. Approximately, when $M \approx N\sqrt{N}$, the diagonal weights dominated in the following way: for $i = 1, \ldots, N$, $\sum_j |w_{ij}| < w_{ii}$. Obviously then all 2^N vectors were stable; the memory therefore became useless.

This diagonal-dominance effect for sufficiently large M admits the following partial explanation. Whenever a pattern $\mathbf{X}$ is presented to the Perceptron-type rule and component i is unstable, i.e. $Y_i \neq_i$, we have

$$w_{ii}(t+1) = w_{ii}(t) + 1 - X_i Y_i = w_{ii}(t) + 2$$

in the case $\eta = 1$. Thus during the storage phase, whenever bit i is unstable, the weight w_{ii} increases by two. The other weights w_{ij} for $i \neq j$ will on the other

[3] Some asymmetry creeps into the rule because of the use of the signum function, which is asymmetric since signum(0) equals 1. This asymmetry makes a difference when the first pattern is present because at that time, the weight matrix $\mathbf{W}$ equals 0.

hand increase, decrease, or remain unchanged when component i or j or both are unstable, depending on X_i, X_j, Y_i, Y_j. Thus since diagonal weights monotonically increase when instabilities are detected while the off-diagonal weights may increase or decrease it seems that eventually the diagonal weights dominate over the off-diagonal ones to the extent that all bits become frozen to their input values. When this happens the memory is useless – all 2^N patterns are stable.

We see thus that storage capacity alone is an inadequate measure of performance. By this measure alone the capacity of the Perceptron-type rule would be $M = 2^N$. This measure does not reveal however that as M increases, the error-correction ability reduces until it eventually becomes zero. A more realistic assessment of capacity is based on evaluating the stability of the memories as well as their attraction domains.

We also tried to ascertain whether and at what M this diagonal-dominance occured in the sparse cases ($p = 0.1, 0.3$) and in the very dense ones ($p = 0.7, 0.9$). We chose $N = 50$ in all these cases, and conducted experiments all the way up to $M = 2000$. The matrix did not become diagonally-dominant in any of the experiments. Thus it seems that in the sparse and very dense cases the matrix becomes diagonally-dominant (if at all) for much larger ratio M/N than in the $p = 0.5$ case.

The second insight is that, by contrast with the Hebb rule, the learning process of the Perceptron-type rule evolved over multiple epochs in somewhat interesting ways. In particular, in some cases, the error did not reduce monotonically over the epochs, though eventually it always went down to zero. Such behavior is not entirely surprising given the roots of the Perceptron-type rule in the Perceptron learning algorithm, where similar behavior is known to occur frequently. Table 2 shows one typical evolution of the rule, on one pattern-set.

The third insight is that generally more learning epochs are required for the Perceptron-type rule for sparse patterns ($p = 0.1$) than in other cases. The only exception is the case of $\langle 60, 200, \cdot, 4\rangle$. For small B, probably the behaviour is again caused by the existence of only the two attractors $(-1)^N$ and its complement under the Hebb rule in these sparse cases. As a result, all patterns tend to the same direction, towards the $(-1)^N$ attractor, and probably much more "effort" is required to properly separate the patterns.

Table 2
Illustration of the non-monotonic evolution of the Perceptron-type rule via one experiment with the parameter setting $\langle M, N, p, B\rangle = \langle 80, 100, 0.5, 1\rangle$

1	2	3	4	5	6	7	8	9
3180	410	107	18	23	2	4	5	0

The upper row denotes the epoch number; the lower one the number of bits remaining unstable in the M library patterns after the end of this epoch.

3.2. The error-correction results

Experiments were conducted to estimate, roughly, the error-correction ability around the stored library vectors and thereby get a more realistic assessment of the capacity. The same sets of patterns as those used for the stability results reported in the previous section were used.

First we consider the case when the M probe vectors were derived by flipping *one* randomly chosen bit in each of the M library vectors. Tables 3 and 4 report the performance of the Perceptron-type and Hebb rules at the pattern- and bit-level respectively, in this case. The library patterns and the probe patterns in both tables were identical. Performance of both rules was evaluated in terms of the patterns that were retrieved after one synchronous recall step starting from the probe vectors.

The results are summarized as follows. First, in terms of both the number of erroneous patterns recalled and the number of erroneous bits in a recalled pattern, the Perceptron-type rule significantly outperformed the Hebb rule for all

Table 3
Error-correction results

M	B	$p = 0.5$		$p = 0.3$		$p = 0.1$	
		#H	#P	#H	#P	#H	#P
20	1	0.05	0.08	1.00	0.13	1.00	0.12
20	2	0.29	0.04	0.98	0.06	1.00	0.06
20	3	0.36	0.04	0.97	0.05	1.00	0.04
20	4	0.47	0.02	0.96	0.05	1.00	0.05
20	5	0.58	0.02	0.91	0.01	0.99	0.02
40	1	0.71	0.30	1.00	0.25	1.00	0.23
40	2	0.83	0.25	0.99	0.14	1.00	0.07
40	3	0.80	0.17	0.99	0.08	1.00	0.04
40	4	0.72	0.11	0.99	0.07	0.99	0.04
40	5	0.66	0.09	0.97	0.06	0.99	0.02
60	1	0.96	0.31	1.00	0.34	1.00	0.28
60	2	0.94	0.21	0.99	0.15	1.00	0.11
60	3	0.87	0.11	1.00	0.13	1.00	0.07
60	4	0.76	0.07	0.99	0.06	0.99	0.04
60	5	0.47	0.07	0.99	0.04	0.99	0.04

Columns labeled #H (#P) denote results where the M library patterns were stored via the Hebb (Perceptron-type) rule. In both cases, #H and #P, the reported numbers are the fraction of the M probe patterns that, after *one synchronous retrieval step*, led to a pattern different from the original library pattern from which the probe was derived. Each number is averaged over the number of trials (10). The M probe patterns were generated by *flipping one randomly chosen bit* from each of the library patterns.

Table 4
Bit error-correction results for exactly the same experiments reported in Table 3: same sets of library patterns, probe patterns, and retrieval conditions (one synchronous step)

M	B	$p = 0.5$		$p = 0.3$		$p = 0.1$	
		#H	#P	#H	#P	#H	#P
20	1	0.05	0.11	29.33	0.14	20.25	0.13
20	2	0.71	0.09	27.34	0.13	20.44	0.12
20	3	1.47	0.13	22.61	0.15	20.73	0.13
20	4	2.68	0.12	21.50	0.22	20.34	0.20
20	5	3.92	0.12	18.75	0.05	20.32	0.10
40	1	1.36	0.17	50.54	0.29	20.41	0.27
40	2	3.47	0.26	45.08	0.30	20.25	0.17
40	3	4.34	0.27	40.38	0.27	20.49	0.15
40	4	4.43	0.23	35.42	0.31	20.44	0.20
40	5	5.12	0.23	32.32	0.32	20.55	0.13
60	1	3.64	0.39	57.22	0.41	20.06	0.33
60	2	5.55	0.46	52.79	0.33	20.43	0.24
60	3	5.47	0.36	48.35	0.41	20.22	0.23
60	4	5.17	0.30	42.75	0.24	20.41	0.16
60	5	4.80	0.37	37.82	0.21	20.52	0.21

The numbers reported in columns #H and #P are the numbers of *bits* that differ between the retrieved pattern and the target library pattern, averaged over the number of patterns (M) and the number of trials (10), for the Hebb and Perceptron-type, rules, respectively.

tried settings of the parameters $\langle M, N, p, B\rangle$ except one: $\langle 20, 200, 0.5, 1\rangle$. On this last setting, the Hebb rule performed slightly better. The performance difference between the two rules is striking in the sparse cases ($p = 0.1, 0.3$). The Perceptron-type rule performed consistently well over all tried p-values (0.5,0.3,0.1), all tried B-values (1,2,3,4,5), and all tried M-values (20,40,60). By contrast, the Hebb rule's performance degraded dramatically in going from the $p = 0.5$ case to the $p = 0.3, 0.1$ cases and in the $p = 0.5, 0.3$ cases as M was increased.

It is worth noting that the Hebb rule results for the 1-bit-distortion case are only slightly worse (in rare cases even better) than those of the stability case (compare Table 4 with Table 1). That is, the number of unstable bits in the retrieved pattern is only slightly more on average when the probe pattern has one bit of error in it than when it has zero bits of error in it. We may take this as a preliminary indication that the Hebb rule is stable in this sense (slight increase in errors in probe do not degrade performance drastically).

In the $p = 0.3, 0.1$ cases this effect is easy to explain. In these cases, the average number of unstable bits in a library pattern is so large that whether one uses a library pattern as a probe or a one-bit distortion of it as a probe matters

little. Indeed it is possible that in both cases the network retrieves an attractor after just one synchronous step and, furthermore, the same one in both cases. This is especially likely in the $p = 0.1$ case where we have noted in previous sections that the network is likely to have formed just one attractor [4] after storage of the M patterns. Therefore all probes would naturally go to this attractor.

The $p = 0.5$ case, though still not surprising, is more difficult to explain. First, the number of unstable bits in both situations (probes being the library patterns or the 1-bit-distorted patterns) is relatively low. Second, the network probably forms a large number of attractors, as argued in a previous section. It may still be the case that for an unstable library pattern, the network usually retrieves the same attractor after one synchronous step whether one starts from this library pattern, or from a 1-bit distortion of it. This reasoning is however at best speculative.

Another noteworthy observation is that, as a function of p, the trend of the performance of the Hebb rule on the 1-bit-distorted probes is very similar to that of the library probes case. The rule performs best on the $p = 0.5$ case, worst in the $p = 0.3$ case, and slightly better than the $p = 0.3$ case on the $p = 0.1$ case. This effect is not surprising, and its explanation is essentially the same as we gave in the library probes case (second and third paragraphs of Section 3.1.1).

A final noteworthy observation is that for fixed M, N and p, the performance of the Perceptron-type rule at the pattern-level improves significantly with increase in B (see Table 3). It is especially interesting that the same does not happen at the bit-level (see Table 4).

This indicates obviously that when B is large, for some 1-bit-distorted probes the distortion increases after one synchronous step and for others it decreases. This is not entirely unreasonable, because when library patterns are highly correlated, the recall behavior from 1-bit-distorted probes can possibly be sensitive to the bit that is distorted, and the library pattern that it is distorted from.

We also conducted experiments with probe patterns derived from library patterns with two randomly-selected bits flipped. The performance of the Hebb rule and the Perceptron-type rule on these probe patterns is qualitatively similar to the results of the previous section, both at the pattern-level and at the bit-level. In absolute terms, the Hebb rule performs nearly as well as in the 1-bit-distortion case. The performance of the Perceptron-type rule degrades by a factor of 1.5 to 2 over its performance on 1-bit-distorted probes. The results are not reported in detail here.

To this point in this section, we have focused on the case $p = 0.5$ and the sparse cases $p = 0.3, 0.1$. As argued in Section 3.1, we would expect the results

[4] Actually two – the second one being its complement.

for the very-dense cases $p = 0.7, 0.9$ to be similar, [5] by symmetry, to the cases $p = 0.3, 0.1$, respectively. We do not report the results here, but this indeed turns out to be the case. Tables 2 and 3 of the conference version of this paper [7] present those results.

3.2.1. Multiple-bit distorted probes

Finally, we conducted experiments in which the probe patterns had d bits of error in them, with d ranging from 5 to $N/2$. The same library patterns as in previous sections were stored. The probe patterns were derived from these library patterns by flipping d bits chosen at random.

Both rules were evaluated at two extremes of correlation of the library patterns: $B = 1$, the uncorrelated case and $B = 5$, a highly correlated one. A range of densities, $p = 0.1, 0.3, 0.5$, were evaluated in these two cases. Finally, the distortion parameter d was varied from 5 to 100. In all experiments, the number of patterns M was fixed to 20 and the pattern-dimension N to 200.

To evaluate the results of this subsection, the following measure is useful. Define *bit-level correction efficiency* as the percentage of the d distorted bits that were corrected, on average, in the retrieved pattern.

The experimental results are presented in Table 5. The main observations are as follows. In the $\langle p, B\rangle = \langle 0.5, 1\rangle$ case, the Perceptron-type rule performed much poorer than the Hebb rule at both the pattern- and the bit-levels, when d ranged from 5 to 30. In all of the remaining $\langle p, B\rangle$ cases, the Perceptron-type rule performed much better than the Hebb rule at both the pattern- and bit-levels, when d ranged from 5 to 30. In virtually all these cases, the bit-level correction efficiency of the Perceptron-type rule was more than 87%.

In addition to the main observations, the following finer ones are also noteworthy.

- The performance of the Perceptron-type rule at both the pattern- and the bit-levels is relatively insensitive to the pattern density p, so long as $p < 0.5$.
- The pattern-level performance superiority of the Perceptron-type rule over the Hebb rule is highest when the library patterns were both sparse and correlated.
- The bit-level performance superiority of the Perceptron-type rule over the Hebb rule is highest when the library patterns were sparse. In particular, it is relatively insensitive to the correlation amount.
- It is interesting, though not surprising, that both the Hebb and the Perceptron-type rule were able to perform significant bit-level correction even on very noisy probes (see the $d = 50, 100$ rows).

[5] They would be identical for the Hebb rule except that the samples would be different in actual experiments. The Perceptron-type rule has a slight asymmetry so they would not be identical even in principle.

Table 5
Error-correction results for the Hebb rule versus the Perceptron-type rule as a function of distortion d in the probe patterns

B	d	$p = 0.5$		$p = 0.3$		$p = 0.1$							
		Patterns		Bits		Patterns		Bits		Patterns		Bits	
		#H	#P	#H	#P	#H	#P	#H	#P	#H	#P	#H	#P
5	5	0.58	0.14	4.07	0.72	0.92	0.04	18.87	0.55	0.99	0.08	20.32	0.42
	10	0.60	0.25	4.45	1.52	0.93	0.19	18.90	1.07	0.99	0.20	20.32	1.07
	15	0.58	0.31	4.65	1.95	0.92	0.30	18.95	1.90	0.99	0.28	20.32	1.62
	20	0.65	0.36	4.75	2.42	0.96	0.34	18.82	1.95	0.99	0.30	20.32	1.82
	30	0.75	0.57	6.42	4.47	0.93	0.52	20.10	3.85	0.99	0.53	20.32	3.97
	50	0.86	0.77	9.60	7.82	0.92	0.78	19.50	8.45	0.99	0.79	20.32	7.55
	100	0.98	0.99	21.62	27.27	0.99	0.99	24.27	30.10	0.99	0.99	20.32	30.70
1	5	0.08	0.28	0.10	0.49	1.0	0.40	29.38	0.61	1.0	0.35	20.25	0.59
	10	0.13	0.44	0.17	1.02	0.99	0.61	28.87	1.25	1.0	0.59	20.25	1.13
	15	0.23	0.64	0.36	1.57	1.0	0.75	28.48	2.10	1.0	0.71	20.25	1.81
	20	0.32	0.76	0.46	2.20	1.0	0.84	28.72	3.17	1.0	0.83	20.25	2.66
	30	0.51	0.92	1.19	4.65	0.98	0.95	28.23	5.57	1.0	0.94	20.25	4.68
	50	0.91	0.99	4.57	10.20	1.0	0.99	27.58	12.13	1.0	0.99	20.25	11.76
	100	1.0	1.0	23.40	33.15	1.0	1.0	30.56	39.76	1.0	1.00	20.25	43.92

The columns labeled *patterns* denote the fraction of the M ($M = 20$) probe patterns that did not recall the original library pattern from which they were derived, for the Hebb (#H) and the Perceptron-type (#P) rules, respectively, averaged over ten trials. The columns labeled *bits* denote the number of bits in a recalled pattern that were different from those in the original library pattern, averaged over the M probe patterns and ten trials.

Interestingly, for the Hebb rule with $p = 0.5$, though the number of bit errors remaining in the retrieved pattern increased slowly as d increased, the correction efficiency in fact improved. For example, when $d = 5$ the correction efficiency was less than 20% while when $d = 50$ it was more than 80%.

Finally, we present two observations on how the results scale with d. These observations draw upon the results presented not only in Table 5 but also the ones for $d = 1$ presented in Table 4 pertaining to $M = 20$ and $B = 1, 5$.

In the density $p = 0.1$ case, with storage under the Hebb rule, the number of bits remaining unstable in the retrieved pattern remains unchanged across all values of d, and across both correlation factors B. This phenomenon is also seen, to a slightly lesser degree, in the $p = 0.3$ case.

By contrast, in all $\langle p, B \rangle$ cases, including those covered in the above paragraph, with storage under the Perceptron-type rule, the number of bits remaining unstable in the retrieved pattern increases roughly linearly with the distortion level d.

4. Potential applications

An associative memory model may be broadly evaluated on two criteria: *stability* – the fraction of library patterns that are stored stably; and *recall* – the fraction of library patterns that are recalled perfectly from their distorted versions as probes (as a function of the number d of distorted bits in the probes).

Though it is very desirable to construct associative memories that meet both criteria very well, it is impossible to construct Hopfield associative memories that have perfect stability *and* good recall when the number of patterns is greater than $0.15n$.

Here we take the somewhat unconventional view that many applications can tolerate poorer performance in one of the criteria, provided the other is sufficiently well met.

The first situation we consider is one in which we relax the second criterion to one involving *approximate recall*. For instance we could replace the recall criterion by the following one: *bit-level correction* – the average number of bit errors in the probes that are corrected (as a function of the original number d of distorted bits in the probes).

An example of an application in which perfect recall may be sacrificed for sufficiently good bit-level error correction is image restoration. If an associative memory model is attractive for other reasons, then one may often tolerate imperfect recall on this application so long as the image is sufficiently well restored.

The second situation is one in which we eliminate the second criterion (recall criterion) entirely. Consider the problem of *detecting* spelling errors in a

document, given a dictionary of valid words stored in the network. The words in the document are input sequentially to the network. In this application recall, i.e. error-*correction*, is not an issue. What *is* an issue is that the first criterion be met perfectly, i.e. all the words be stored stably. In other words, the network should not detect a correct word as an error. From this point of view, the Perceptron-type rule is clearly attractive since it ensures that all library patterns are stored stably.

Of course, it may be that if the number of dictionary words is very large, the Perceptron-type rule may also store many spurious memories. In this case, many misspelt words might go undetected as errors. To what extent this happens is an empirically testable question. Here we briefly argue that this is less of a problem than our experiments in this paper may indicate.

First, one may adopt a *sparse* encoding of dictionary words as binary vectors to alleviate this problem. One such successful encoding is in [8] where each character was encoded in 62 bits (26 for lower-case, 26 for upper-case, 10 for the 10 digits). This encoding is less efficient than the standard ASCII one. However this is not a major problem. For example, to store an arbitrary collection of ten letter-words stably in a network one would employ 620 neurons, a managable number, via this encoding. Second, dictionary words are highly correlated. Our experiments in this paper suggest that the combination of sparseness and correlatedness of the library patterns is expected to make the Perceptron-type rule work well.

The Hebb rule on the other hand does not appear to be suited to this problem at all, because its stability degrades rather quickly as the number of library patterns is increased, especially when the patterns are sparse or correlated.

5. Conclusions

This paper has conducted an extensive experimental study of a Perceptron-type learning rule for the Hopfield associative memory, and compared its performance with that of the Hebb rule. The rules were evaluated and compared on a wide range of conditions on the library patterns: the number of patterns M, the pattern-density p, and the amount of correlation B. Performance was evaluated on the criteria of stability of the library patterns, and error-correction performance on noisy probe patterns, measured at both the pattern- and bit-levels. The Perceptron-type rule was found to be 100% correct on the library pattern stability criterion alone. By contrast, the performance of the Hebb rule on this criterion was found to degrade rapidly as M was increased. When between $0.1N$ and $0.3N$ patterns were stored, the Perceptron-type rule significantly outperformed the Hebb rule on the error-correction criteria in all cases except when the $0.1N$ patterns were generated uniformly at random ($p = 0.5$, $B = 1$). The error-correction performance of the Perceptron-type rule

remained roughly constant over various degrees of sparseness and correlation of the library vectors, while that of the Hebb rule degraded drastically as the sparseness or the correlation or both were increased.

The sample size of 10 is a bit small. To examine the impact of this, we repeated several of the experiments. In the cases where the Hebb rule performed poorly, we found the variance (of the number of unstable bits of the library patterns for example) to be negligible in comparison to the mean value of the same quantity. In situations where the Hebb rule performed reasonably well, for example in row 1 of Table 1 and the first column containing #u ($p = 0.5$ case), we found that the variance was not negligible in comparison to the mean value of this column. However this variance was negligible in comparison with means of other columns, for example the ones in which the patterns were more sparse ($p = 0.3, 0.1$). Thus the means seemed to capture reasonably well the trends across the data sets (varying density $p = 0.5, 0.3, 0.1$, varying block-size $B = 1, 2, 3, 4, 5$, varying number of library patterns $M = 20, 40, 60$).

One reason for our choice of this sample size is the large number of experiments that were performed. The perceptron-type rule used up a fair bit of time during training, since the library patterns were needed to be presented to it multiple times.

The most important question that remains open in our studies is to theoretically analyze the error-correction performance of the Perceptron-type rule. One may attempt to obtain results on this rule of the kind known for the Hebb rule [3].

References

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

[2] J. Hertz, A. Krogh, R.G. Palmer, Introduction to the Theory of Neural Computation, Addison–Wesley, Reading, MA, 1991.

[3] R.J. McEliece, E.C. Posner, E.R. Rodemich, S.S. Venkatesh, The capacity of the Hopfield associative memory, IEEE Transactions on Information Theory 33 (1987) 461–482.

[4] J. Mańdziuk, Improving performance of the bipolar Hopfield network by supervised learning, in: Proceedings of the World Congress on Neural Networks, San Diego, CA, September 1996, pp. 267–270.

[5] Y. Kamp, M. Hasler, Recursive Neural Networks for Associative Memory, Wiley, New York, 1990.

[6] D. Liu, Z. Lu, Associative memory design via perceptron learning, in: Proceedings of the IEEE International Conference on Neural Networks, Houston, 1996, pp. 1172–1177.

[7] J. Mańdziuk, A. Jagota, Experimental study of Perceptron-type online local learning rule for Hopfield associative memory, in: Proceedings of Second International Conference on Computational Intelligence and Neuroscience, Research Triangle Park, NC, March 1997.

[8] A. Jagota, Contextual word recognition with a Hopfield-style net, Neural, Parallel, and Scientific Computations 2 (2) (1994) 245–271.