Comparison Between Sparsely Distributed Memory and Hopfield-type Neural Network Models

James D. Keeler

December, 1986

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 86.31

RIACS

Research Institute for Advanced Computer Science

Comparison Between Sparsely Distributed Memory and Hopfield-type Neural Network Models

James D. Keeler

University of California, San Diego, Physics Department and
Institute for Nonlinear Science B-019, La Jolla, CA 92093

(December 1986)

ABSTRACT: The Sparsely Distributed Memory (SDM) model (Kanerva, 1984) is compared to Hopfield-type neural-network models. A mathematical framework for comparing the two is developed, and the capacity of each model is investigated. The capacity of the SDM can be increased independently of the dimension of the stored vectors, whereas the Hopfield capacity is limited to a fraction of this dimension. However, the total number of stored bits per matrix element is the same in the two models, as well as for extended models with higher order interactions. The models are also compared in their ability to store sequences of patterns. The SDM is extended to include time delays so that contextual information can be used to recover sequences. Finally, it is shown how a generalization of the SDM allows storage of correlated input pattern vectors.

Introduction

Hopfield (1982) presented an autoassociative memory model based on a network of highly interconnected two-state threshold units ("neurons"). He showed how to store randomly chosen patterns of 0s and 1s in the network by using a Hebbian (Hebb, 1949) learning algorithm. He was also able to show that for symmetric connections between the units, the dynamics of this network is governed by an energy function that is equivalent to the energy function of an Ising model spin-glass (Kirkpatrick & Sherrington, 1979). The two-state units used in the Hopfield model date back to McCulloch and Pitts (1943), and models of this type are currently known as "neural-network models" (or "connectionist models" or "artificial neural systems"). Similar models have been investigated by Amari (1971), Anderson *et. al.* (1977), Kohonen (1980), Little and Shaw (1978), and Nakano (1972).

The Hopfield neural-network model is attractive for its simplicity and its ability to function as a massively parallel, autoassociative memory. Nevertheless, a number of limitations of the Hopfield model have been pointed out. First of all, the storage capacity (the number of memory patterns that can be stored in the network) is limited to a fraction of the number of processing elements (McEliece *et al.*, 1986). Second, the standard Hopfield model is unsuccessful at storing temporal sequences of memory patterns (Hopfield, 1982). Third, as a model of the brain, it is unrealistic due to the requirement of symmetric connections between the units. Finally, it is quite limited in its ability to store sets of correlated patterns.

Kanerva (1984) introduced a memory model known as Sparsely Distributed Memory (SDM) (or Sparse, Distributed Memory), that is not restricted by the limitations listed above. Although independently discovered, Kanerva's model is basically equivalent in mathematical form to a model of the cerebellar

cortex introduced by Marr (1969) and to the Cerebellar Model Arithmetic Computer (CMAC) introduced by Albus (1971). I will follow Kanerva's description and denote the model SDM. The SDM model uses nonsymmetric matrices in a two-stage system to store patterns, and it can function as an autoassociative memory, a heteroassociative memory, or a sequential-access memory.[1]

In the following I develop a mathematical framework for comparing the SDM and the Hopfield model. I then analyze the storage capacity of both models and their ability to store sequences of patterns. I also show how the Hopfield model can be thought of as a special case of a mathematical extension of the SDM. I then compare the SDM to a few other models that have been proposed to alleviate some of the limitations of the Hopfield model. Finally, I show how the SDM can be used to store correlated sets of patterns.

Hopfield Model

In this section, I briefly review the Hopfield model in its discrete form (Hopfield, 1982), and introduce the mathematical formalism that will be used in discussing the SDM. The processing elements (units) in the Hopfield model are simple two-state threshold devices: The state of the i^{th} unit, u_i, is either + 1 (firing at the maximum rate) or -1 (not firing). Consider a set of n such units with the connection strength from the j^{th} unit to the i^{th} given by T_{ij}. The net input to the i^{th} unit from all of, h_i, the other units is given by

$$h_i = \sum_{j=1}^{N} T_{ij} u_j. \tag{1}$$

[1] An autoassociative memory is one that associates a pattern with itself and is synonymous with a content-addressable memory, whereas a heteroassociative memory is one that associates one pattern with another, and a sequential-access memory yields a temporal sequence of patterns.

The state of each unit is updated asynchronously (at random) according to the rule

$$u_i \leftarrow g(h_i) \ , \tag{2}$$

where for the discrete model g is a simple threshold function

$$g(x) \equiv \begin{cases} +1 \text{ if } x > 0 \\ \textit{unchanged} \text{ if } x = 0 \\ -1 \text{ if } x < 0 \end{cases} . \tag{3}$$

In this formulation, the dynamics is not deterministic because each unit is updated at random times.

The state of all of the units at a given time can be thought of as a firing pattern of the units. This pattern is just an n dimensional vector **u** whose components are $\pm$ 1. Each different pattern can be represented as a point in the n dimensional state space of the units, and there are 2^n distinct points in this space. The goal here is to store a few "memory" patterns in this space as stable fixed points of the dynamical system. This allows the system to function as an accretive autoassociative memory: When the system is given as input a pattern that is close to one of the stored patterns it will relax to that stored pattern (a fixed point).

Suppose we are given M patterns (strings of length n of $\pm$ 1s) that we wish to store in this system. Denote these M memory patterns as pattern vectors: $\mathbf{p}^\alpha = (p_1^\alpha, p_2^\alpha, \ldots, p_n^\alpha)$, $\alpha = 1,2,3,\ldots,M$. For example, $\mathbf{p}^1$ might look like $(+1,-1,+1,-1,-1,\ldots,+1)$. One method of storing these patterns is the so-called Hebbian learning rule: Start with $T_{ij} \equiv 0$, accumulate the outer products of the pattern vectors

$$T \leftarrow T + [\mathbf{p}^\alpha \times \mathbf{p}^\alpha], \quad \text{for } \alpha = 1,2,3,\ldots,M, \tag{4}$$

where $\times$ denotes the outer product of the two vectors, and set the diagonal elements $T_{ii} = 0$. The resulting matrix, after learning all of the patterns, is given

by

$$T_{ij} = \sum_{\alpha=1}^{M} [p_i^{\alpha} p_j^{\alpha} - \delta_{ij}], \tag{5}$$

where the δ_{ij} is necessary[2] to set the diagonal terms to zero. It is shown below that these stored memory patterns, $\mathbf{p}^{\alpha}$, will be attracting fixed points of Equation (2) provided that certain conditions on the number of patterns and their statistical properties are satisfied.

To see why this rule for storing the patterns works, suppose that we are given one of the patterns, $\mathbf{p}^{\beta}$, say, as the initial configuration of the units. First I will show that $\mathbf{p}^{\beta}$ is expected to be a fixed point of Equation (2), then I will show that $\mathbf{p}^{\beta}$ is expected to be an attracting fixed point. I present the analysis in detail here because this same analysis is carried out again for the more complicated cases of the SDM.

Insert Equation (5) for T into (2). The resulting expression for the net input to the i^{th} unit becomes

$$h_i = \sum_{\alpha=1}^{M} [\sum_{j=1}^{n} (p_i^{\alpha} p_j^{\alpha} - \delta_{ij}) p_j^{\beta}]. \tag{6}$$

The important term in the sum on α is the one for which $\alpha = \beta$. This term represents the "signal" between the input $\mathbf{p}^{\beta}$ and the desired output. The rest of the sum represents "noise" that comes from all of the other stored patterns. Hence, separate the sum on α into two terms: the single term that has $\alpha = \beta$ and the rest of the sum, $\alpha \neq \beta$. The expression for the net input becomes

$$h_i = signal_i + noise_i, \tag{7}$$

where

$$signal_i = p_i^{\beta} \sum_{j=1}^{n} p_j^{\beta} p_j^{\beta} - M p_i^{\beta} \tag{8}$$

and

---

[2] This is a Kronecker δ: $\delta_{ij} = 0$ for $i \neq j$; $\delta_{ij} = 1$, for $i = j$.

$$noise_i = \sum_{\alpha \neq \beta}^{M} \sum_{j=1}^{n} p_i^{\alpha} p_j^{\alpha} p_j^{\beta}. \tag{9}$$

Note that this noise term is dependent only on the stored patterns; there is no external "temperature" causing the noise in this model.

Summing on j in (8) yields

$$signal_i = (n-M)p_i^{\beta}. \tag{10}$$

Since $n-M$ is positive for $n > M$, the signs of the signal term and p_i^{β} will be the same. Thus, if the noise term were exactly zero, the signal would give the same sign as p_i^{β} with a magnitude of $n-M$, and $\mathbf{p}^{\beta}$ would be a fixed point of Equation (2). It is easy to see that $\mathbf{p}^{\beta}$ would be an attracting fixed point (if the noise were zero) by considering an initial condition that is slightly displaced from $\mathbf{p}^{\beta}$. If the initial condition differed by k bits from $\mathbf{p}^{\beta}$, the signal would still give the same sign as p_i^{β}, with strength $n-M-2k$. Thus, if $k < (n-M)/2$ the signal would give the proper sign, and $\mathbf{p}^{\beta}$ would be an attracting fixed point.

If the stored patterns are chosen to be orthogonal vectors, the noise term (7) would be exactly zero. However, if the patterns are chosen at random, they are only pseudo-orthogonal, and the noise term may be nonzero. Its expected value, however, will be zero, $< noise > = 0$, where $< >$ indicates statistical expectation, and its variance will be

$$\sigma^2 = n(M-1). \tag{11}$$

Hence, the probability that there will be an error on recall of p_i^{β} is given by the probability that the noise is greater than the signal, as shown in Figure 1. For n large, the noise distribution is approximately Gaussian, and the probability that there is an error in the i^{th} bit is

$$p_e = \frac{1}{\sqrt{2\pi}\sigma} \int_{n-M}^{\infty} e^{-x^2/2\sigma^2} dx. \tag{12}$$

Thus, for $p_e << 1$ (M not too large), the stored patterns should be attracting

fixed points.

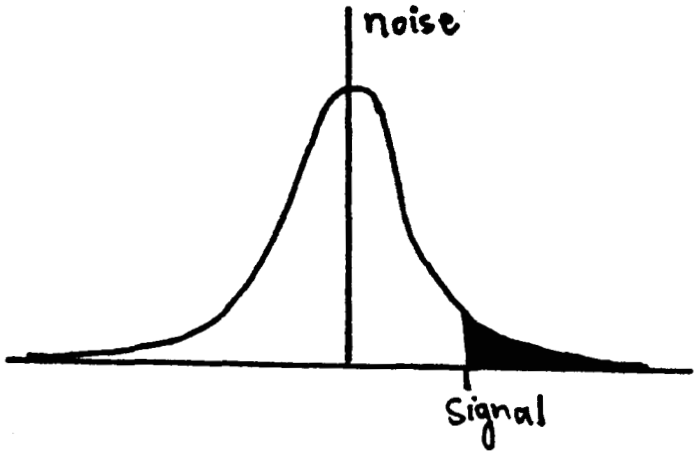

Figure 1. The distribution of noise as described by Equation (9). A bit will be in error when the noise is greater than the signal. The probability of error is equal to the area shaded under the curve.

Description of Sparsely Distributed Memory

In this section, I give a qualitative description of the Sparsely Distributed Memory as formulated by Kanerva (1984). The description is based on parallels between a conventional computer memory and the SDM. I develop the mathematical description that will ease the comparison to the Hopfield model in the next section.

The random-access memory of a conventional computer is an array of storage locations. Each storage location is identified by a number (the address of the location) that specifies the position of the location in the array. Information is stored in the location as a binary word (the contents of the location). Note that the address of the storage location and the contents of that location need not have anything in common. Data are written into a location by giving both the address of the location and the data to be written. The address points to the proper location, and the contents of that location are replaced by the given data. Similarly, data are read from a location by specifying the address of the location, and the contents of that location are read out as the data. The

number of locations that can be accessed in this manner is determined by the length of the input address. If the address is a binary word of length n, then 2^n locations can be accessed. For example, if $n = 16$, then $2^{16} = 64K$ words of memory can be accessed; these words could be 32 bit, 64 bit, or any other size.

The set of 2^n distinguishable n-bit addresses is called the address space (this space is identical to the state space described above). Consider an extension of the conventional computer memory to a memory with very large addresses. For n moderately large, the number of possible addresses becomes astronomical. Indeed, for $n = 1000$, 2^n is larger than the number of atoms in the known universe. Obviously, there is no way of associating all, or even a relatively small fraction, of these addresses with physical storage locations. How can one construct a memory using these large addresses in a sensible manner? Kanerva's answer is as follows: Pick at random m addresses to be associated with physical storage locations (m might be a million to a billion). Because m is small compared to 2^n, these randomly chosen addresses represent a set of storage locations that is sparsely distributed in the address space.

To function as a memory, this system should be able to write and read data. To write, we need as input both the address and the data itself (just as in a conventional computer memory). In the SDM, the address size and the data-word size are allowed to be different. However, for the SDM to function as a autoassociative memory and to compare it to the Hopfield model, I consider only the case where the data-word is the same size as the address; both the address and the data are n-bit vectors of $\pm$ 1s as considered above.

Given an address, where are the corresponding data written? The input address is quite unlikely to point to any one of the m randomly chosen storage locations. However, some of the storage locations are closer to the given

address than others. In the SDM, the data are written into a few selected storage locations that have addresses close to the input address. The selection rule is: select all locations whose addresses are within a Hamming distance D of the input address.[3] If we view these n-bit addresses as points in an n-dimensional address space, the selected locations will lie within a (hyper)sphere of Hamming radius D centered at the input address (see Figure 2). The data

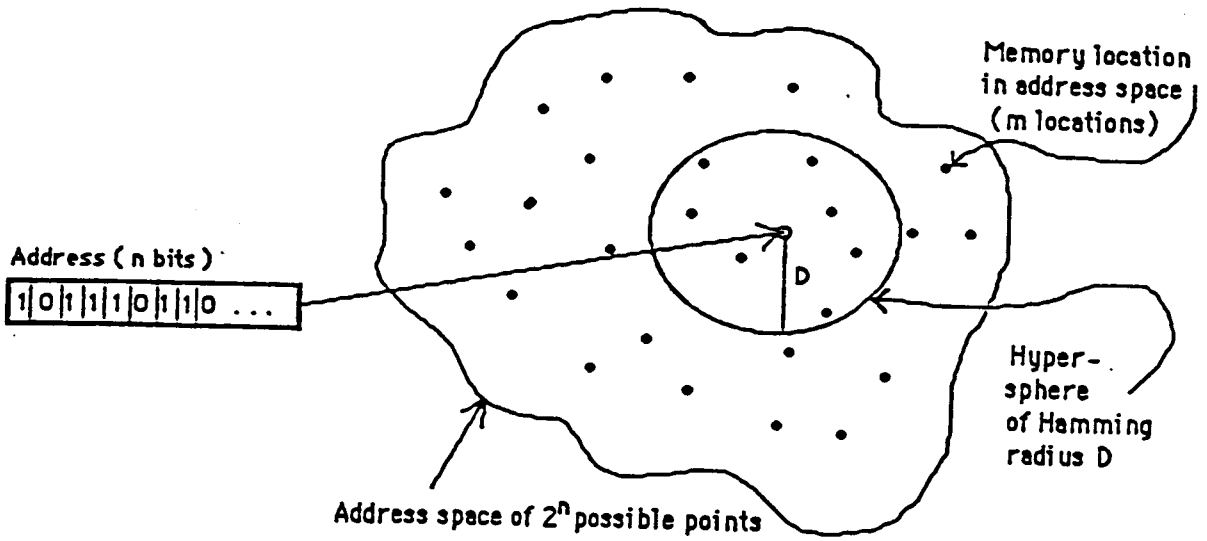

Figure 2. This qualitative picture of address space shows storage locations in the address space, an input address to read from or write into memory, and the Hamming (hyper)sphere containing all selected locations.

are written into every storage location within this sphere. This is why we say that the information is *distributed* over all the selected storage locations. However, the write procedure is a little more complicated than for a conventional computer. Instead of just replacing the old contents of a storage location with the new data, the new data vector is added to the previous contents. Thus, each of the storage locations in the SDM is actually a set of n counters. The reason is that we may wish to write two or more data vectors into any given storage location, because the spheres chosen by two input addresses may

[3] The Hamming distance of two n-bit binary vectors is simply the number of bits at which the two vectors differ. The Euclidean distance is proportional to the square-root of the Hamming distance.

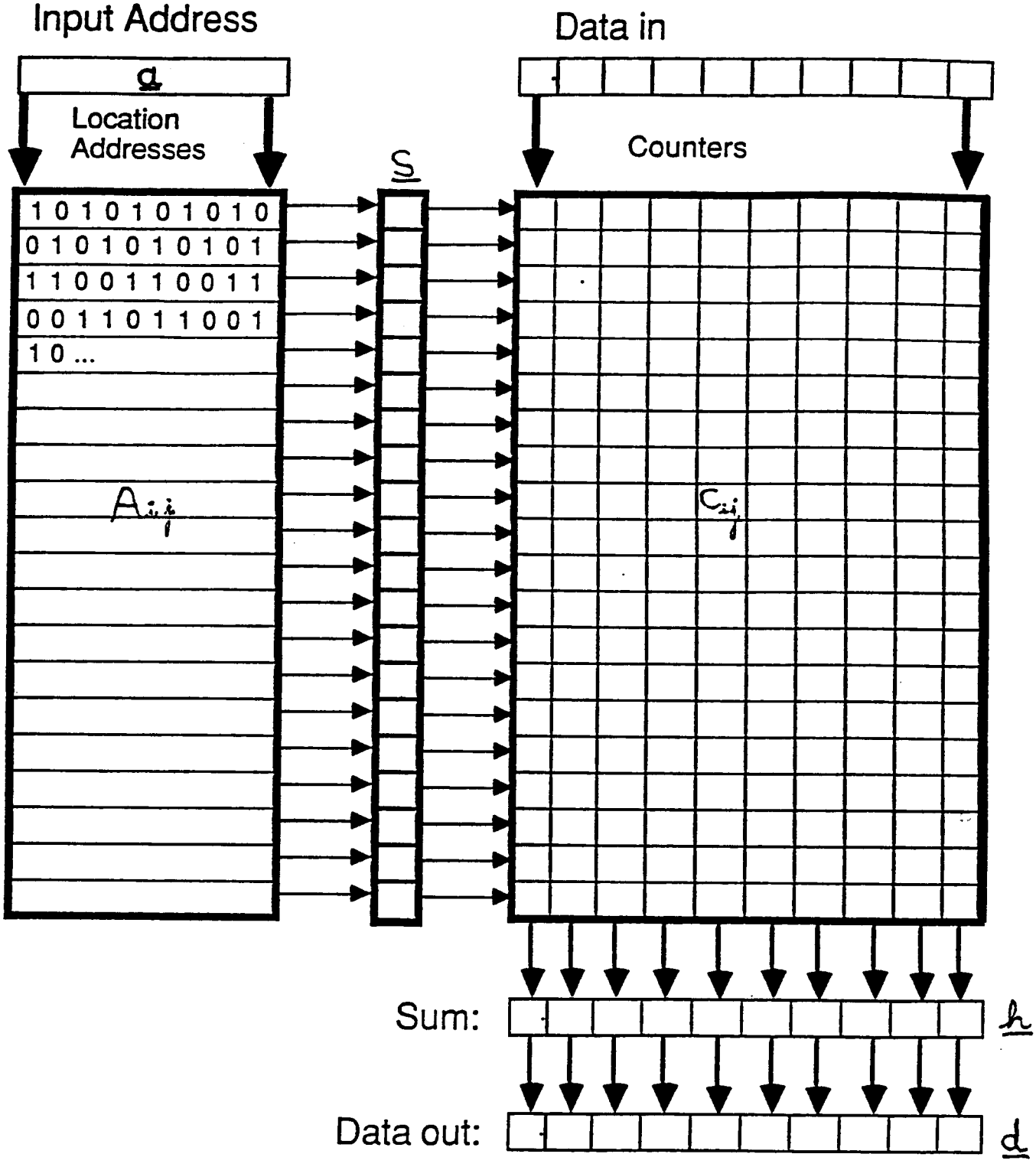

Figure 3. A schematic picture of the functioning of the SDM. The input address comes in at the top as the vector **a**. This address is compared to all of the addresses of the storage locations. These addresses are contained in the matrix $\boldsymbol{A}$ with elements A_{ij}. The selected locations have their select bit set to 1 in the vector **s**. The data are written into the selected locations. The contents of the j^{th} counter of the i^{th} location is given by the matrix element C_{ij}. In a read operation, the contents of the selected locations are added together to give the field **h**. Finally, this field is thresholded to yield the output data **d**.

overlap.

To read from the SDM, the address of the desired data is given and compared to the m addresses of the storage locations. Again, select all locations whose addresses lie within a Hamming sphere of radius D of the given address. The values of these selected locations are added together in parallel to yield n sums (see Figure 3). These sums are thresholded at zero giving a $+1$ in the i^{th} bit if the i^{th} sum is greater than zero, and a -1 if the i^{th} sum is less than zero. Note that this is a statistical reconstruction of the original data word. The output data should be the same as the original data as long as not too many other words have been written into the memory.

This qualitative description of the SDM may seem quite different than the Hopfield model described above, but the read-write rule is just a generalized Hebbian learning rule as shown below.

Layered Network Description

The Hopfield model can be viewed as a two-layer neural network with each layer having n threshold units. The connections between the two layers of units is given by the symmetric $n \times n$ matrix $\boldsymbol{T}$. For the standard autoassociative Hopfield model, the output of the second layer is fed back into the first layer, effectively making the system a one-layer network. The matrix elements of $\boldsymbol{T}$ are given by the Hebbian learning rule. Each unit in the system is updated asynchronously, independently of the other units.

The SDM, on the other hand, can be viewed as a three-layer network (Figure 4). The first layer is the n input units (the input address, **a**), the middle layer is a layer of m hidden units (the selected locations, **s**, and the third layer is the n output units (the data, **d**). The connections between the first layer and the second layer are fixed, random weights and are given by the $m \times n$

matrix A. The connections between the hidden units and the output layer are given by the $n \times m$ connection matrix C. These matrix elements are modified by a Hebbian learning rule. This connection matrix C is analogous to the connection matrix T of the Hopfield model. The output layer can be fed back into the input layer effectively making the SDM a two layer network.

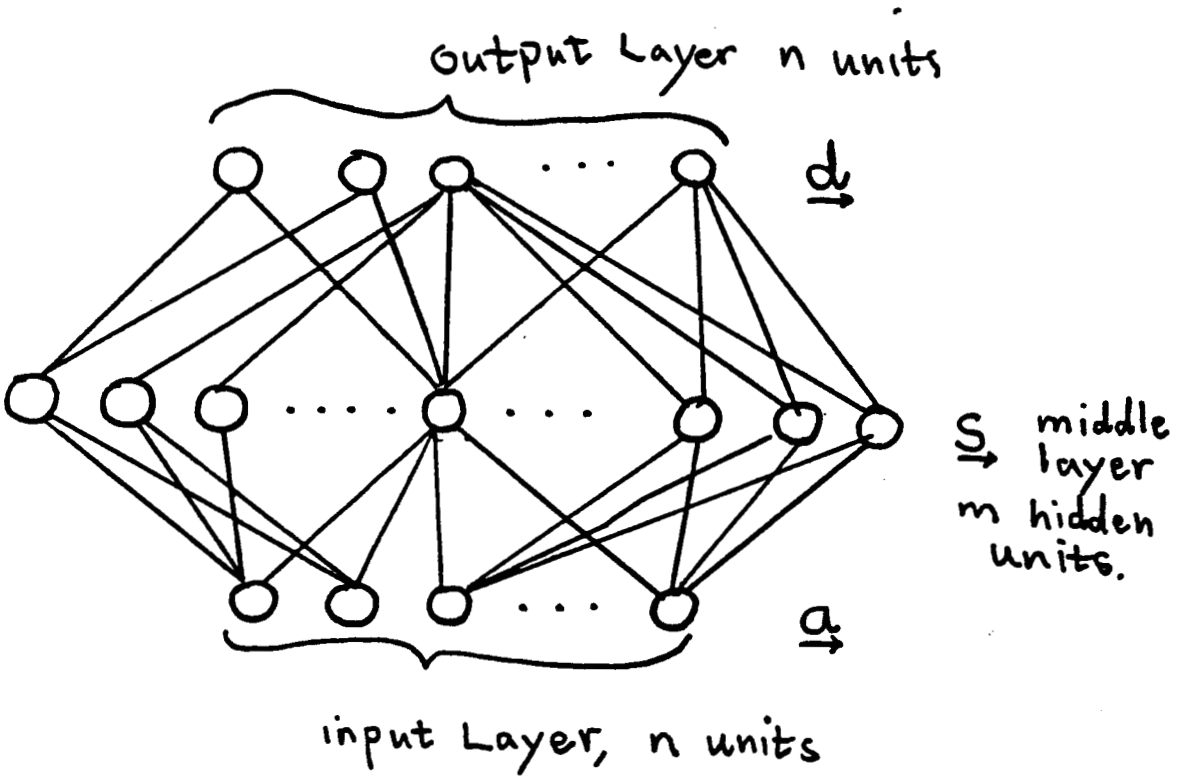

Figure 4. A schematic diagram of the SDM as a three-layer network processing units. The input pattern is the bottom layer **a**, a set of n units whose values are $\pm$ 1. The middle layer is a set of m hidden units that take on the value 0 or 1 and are denoted by the select vector **s**. In the SDM, m is assumed to be much larger than n. The weights between the first two layers is the matrix A. The third layer is the output data **d**, also a set of n units of $\pm$ 1. The connections between the second and the third layer are given by the matrix C.

The SDM typically has $m >> n$, so that the first layer of the network effectively blows the n bit pattern up into a large m dimensional space. Most of the units in the hidden layer are not active. Hence, the hidden unit layer can be viewed as a type of sparse coding of the input pattern (Willshaw *et al.* , 1969). A hidden unit is turned on if the net input to that unit is above a cer-

tain threshold. The input pattern **a** is a vector of length n. The weights to the k^{th} hidden unit from the input units is just the k^{th} row of the A matrix and can be thought of as an n dimensional vector $\mathbf{a}_k$. If both the input pattern and the weights to each hidden unit are binary vectors, then only the hidden units whose weights $\mathbf{a}_k$ are within D Hamming units of the input **a** will be activated; the activated hidden units will be those for which $\mathbf{a}_k$ is close to **a** (the dot product, $\mathbf{a}_k \cdot \mathbf{a}$ is above the threshold).

Mathematical Formulation of SDM

In this section, I will use our earlier mathematical formalism to describe and analyze the SDM. Vector notation will be used here because, unlike the Hopfield model, the units in the SDM are updated synchronously (all units are updated at the same time). Let the addresses of the storage locations be given in terms of vectors of length n of $\pm$ 1s. Since there are m storage locations in the system, there will be m corresponding addresses of those locations. Let A be the matrix whose k^{th} row is the address of the k^{th} storage location (see Figure 3). Hence, A will be an $m \times n$ matrix of random $\pm$ 1s. Let the input address be **a**, a vector of $\pm$ 1s of length n. The contents of the storage locations will be denoted by a counter matrix, C, an $n \times m$ matrix of integers. In a read or write operation, the input address **a** is compared to the addresses of the m locations and the locations within the Hamming sphere are selected. Denote this selected set by a select vector **s**, an m dimensional vector that has 1s at the selected locations and 0s everywhere else.

Given an input address **a**, the selected locations are found by

$$\mathbf{s} = \theta_D(A\,\mathbf{a}), \tag{13}$$

where θ_D is the Hamming-distance threshold function giving a 1 in the kth row if the input address **a** is at most D Hamming units away from the kth address

in $\boldsymbol{A}$ and a 0 if it is further than D units away, *i.e.*,

$$\theta_D(\mathbf{x})_i = \begin{cases} 1 & \text{if } \tfrac{1}{2}(n-x_i) \le D \\ 0 & \text{if } \tfrac{1}{2}(n-x_i) > D \end{cases}. \tag{14}$$

The select vector $\mathbf{s}$ is mostly 0s with an average of δm 1s, where δ is some small number dependent on D; $\delta << 1$ ($\delta \approx 1/\sqrt{m}$ in a typical implementation). Explicitly, δ is given by

$$\delta = \frac{\int_0^D \rho(x)dx}{\int_0^n \rho(x)dx}, \tag{15}$$

where $\rho(x)$ is the probability distribution of the number of points of distance x in the address space from a given point. In this case, $\rho(x)$ is just a binomial distribution centered at $\frac{n}{2}$.

Once the input is presented at the first layer and the locations are selected in the middle layer, the data is output in the final layer. The net input, $\mathbf{h}$, to the final layer is the sum of the selected locations, which can be simply expressed as the matrix product of $\boldsymbol{C}$ with $\mathbf{s}$:

$$\mathbf{h} = \boldsymbol{C}\mathbf{s}. \tag{16}$$

The output data is then given by

$$\mathbf{d} = \mathbf{g}(\mathbf{h}), \tag{17}$$

where $\mathbf{g}$ is the vector analog of the threshold function of Equation (3),

$$\mathbf{g}(\mathbf{x})_i \equiv \begin{cases} +1 & \text{if } x_i > 0 \\ \textit{unchanged} & \text{if } x_i = 0 \\ -1 & \text{if } x_i < 0, \end{cases} \tag{18}$$

How are patterns stored in the SDM? To store a pattern, we must have both the input pattern (address) and the output pattern (data). Suppose we are given M input-output pairs to store in the SDM: $(\mathbf{a}^1,\mathbf{d}^1),(\mathbf{a}^2,\mathbf{d}^2), \cdots ,(\mathbf{a}^M,\mathbf{d}^M)$. This input-output-pair notation is general and allows association between any

two sets of patterns. For example, in an autoassociative memory such as the Hopfield model, $\mathbf{d}^\alpha \equiv \mathbf{a}^\alpha$, whereas a sequential memory is created by setting $\mathbf{d}^\alpha \equiv \mathbf{a}^{\alpha+1}$.

To store the patterns, form the outer product of the data vectors and their corresponding select vectors,

$$C = \sum_{\alpha=1}^{M} \mathbf{d}^\alpha \times \mathbf{s}^\alpha, \tag{19}$$

where the select vector is formed by the corresponding address

$$\mathbf{s}^\alpha = \theta_D(A\,\mathbf{a}^\alpha). \tag{20}$$

The storage algorithm (19) can be thought of as a generalized Hebbian learning rule similar to that of (5).

Now that this mathematical machinery for the SDM has been built, the rest of the analysis proceeds analogously to the discussion of the Hopfield model. First I will show that data written at a given address can be read back from that address. This is analogous to showing that the stored patterns are fixed points of the Hopfield model.

Suppose that the M address-data pairs have been stored according to Equation (19). Here I show that if the system is presented with $\mathbf{a}^\beta$ as input, the output will be $\mathbf{d}^\beta$. Setting $\mathbf{a} = \mathbf{a}^\beta$ in Equation (13) and separating terms as before, the net input (16) becomes

$$\mathbf{h} = \mathbf{signal} + \mathbf{noise}, \tag{21}$$

where

$$\mathbf{signal} = \mathbf{d}^\beta(\mathbf{s}^\beta \cdot \mathbf{s}^\beta) \tag{22}$$

and

$$\mathbf{noise} = \sum_{\alpha \neq \beta}^{M} \mathbf{d}^\alpha(\mathbf{s}^\alpha \cdot \mathbf{s}^\beta). \tag{23}$$

Recall that the select vectors have an average of δm 1s and the remainder 0s.

Hence, the expected value of the signal term is

$$< \text{signal} > = \delta m \, \mathbf{d}^{\beta}. \tag{24}$$

Since δm is positive, the read data will be $\mathbf{d}^{\beta}$ as conjectured (assuming negligible noise).

Assuming that the addresses and data are randomly chosen, the expected value of the noise is zero, with variance

$$\sigma^2 = (M-1)\delta^2 m\,(1+\delta^2(m-1)). \tag{25}$$

The probability of incurring an error in any particular bit of a recalled pattern is

$$p_e = \frac{1}{\sqrt{2\pi}\,\sigma}\int_{\delta m}^{\infty} e^{-x^2/2\sigma^2}dx. \tag{26}$$

The noise term has the same characteristics whether the addresses themselves or other patterns are stored as data. Thus, the SDM can serve as a heteroassociative memory as well as an autoassociative memory. Since the length of the data word is equal to that of the address, the output pattern can be fed back in as a new input address. Iterating in this manner on autoassociative patterns will cause the system to converge onto the stored pattern in much the same manner as for the Hopfield model. Since the connection matrix is not symmetric, there is no energy function governing the behavior of the system. Nevertheless, the above analysis shows that the stored data can be retrieved with high probability as long as not too many patterns have been stored.

Note that if θ_D is allowed to be a generalized vector function rather than the special function described above, then the Hopfield model results from setting $m = n$ and $\theta_D(A\,\mathbf{a}) = \mathbf{a}$, *i.e.*, $\theta_D(A\ ) = \mathbf{1}$, the identity operator. Thus, the Hopfield model may be viewed as a special case of a simple generalization of the SDM . However, no choice of D and A in the standard SDM model yields $\theta_D(A\ ) = \mathbf{1}$.

Capacity of the SDM and the Hopfield Model

An important question to ask about these models is how many patterns can they store. To answer this question, we can restrict our discussion to autoassociative memories. The number of patterns that can be stored in the memory is known as its capacity. There are many ways of defining capacity. One definition is the so-called epsilon capacity, which is simply the number of patterns that can be recalled within epsilon error of the stored pattern vectors. It has been shown theoretically that the epsilon capacity, c_ε, for the Hopfield model is $c_\varepsilon = n/2\log n$ for $\varepsilon > 0$ and $c_0 = n/4\log n$ (McEliece, *et al.*, 1986) for $\varepsilon = 0$ in the limit as $n \to \infty$ (where log is to the base e).

The epsilon capacity may characterize the Hopfield model well for large n, but for small n, other rules have been used. The folklore about the capacity of the Hopfield model is that spurious memories start to appear when the number of stored patterns is about $0.14n$ and the performance of the system is severely degraded beyond that point. This view has been supported by numerical investigations (Hopfield, 1982) and theoretical investigations (Amit *et al.*, 1985).

The capacity of the SDM is not as well understood. First of all, there are more free parameters in the SDM than in the Hopfield model. In the SDM, the capacity is a function of n,m, and D. Kanerva shows the capacity to be proportional to m for fixed n and the best choice of Hamming distance, D. The point here is that the capacity for the SDM increases with m independently of n. Preliminary numerical investigations of the capacity of the SDM by Cohn, Kanerva, and Keeler (1986) have shown that spurious memories appear at approximately $0.13m$, and that the spurious memories can be fixed points, cycles, or "chaotic" wanderings throughout a portion of the pattern space. Hence, a good rule of thumb for the capacity of the SDM is $0.13m$ (for $n \approx 100-1000$).

For a given n, the SDM can store more patterns than the Hopfield model by increasing m. However, how does the total stored information (total number

of bits) compare in the two models? The models can have different n with the same number of stored bits, so that the vector capacity does not yield a fair comparison of the total information capacity of the networks. Therefore, I choose to use a different definition of capacity that gives a more equitable comparison.

The capacity used here has its roots in information theory (Shannon, 1948) and measures the total information stored in the network. This is basically the same definition used by Willshaw *et al.* (1969). Define the *bit capacity* as the number of bits that can be stored in a network with fixed probability of getting an error in a recalled bit. The bit capacity of each system can be investigated by setting $p_e = constant$ in Equations (12) and (26). Setting p_e to a constant is tantamount to keeping the signal- to-noise ratio (fidelity) constant. Hence, the bit capacity of these networks can be investigated by examining the fidelity of the models as a function of n, m, and M. From Equations (10) and (11) the fidelity of the Hopfield model is given by

$$R_{Hop} = \frac{n - M}{\sqrt{(M-1)n}}. \tag{27}$$

This formula yields fixed probability of getting an error in a stored bit for constant R. For example, given $R = 3$, the noise is greater than the signal at 3 standard deviations, and the probability of getting a bit error is ≈ 0.0055. From Equations (24) and (25), the fidelity for Kanerva's model is given by

$$R_{SDM} = \frac{\sqrt{m}}{\sqrt{(M-1)(1+\frac{\zeta^2}{m}(1-1/m))}}, \tag{28}$$

where $\zeta = \delta m$ is the signal strength (the number of selected locations). For $\zeta = constant$ ($\zeta \approx \sqrt{n}/R$), the fidelity for the SDM increases monotonically with m and is approximately the same as the fidelity for the Hopfield model in the limit of large m when $m = n$.

Before looking at the total stored information, we can get an approximate expression for the vector capacity by solving for M in these equations. For the Hopfield model, the capacity in the limit $R >> 1$, $M >> 1$ is given by

$$M = \frac{n}{R^2}(1 - \frac{2}{R^2}), \tag{29}$$

whereas the corresponding expression for the SDM (for large m, M) yields

$$M = \frac{m}{R^2}(1-\zeta^2/m). \tag{30}$$

This last formula shows that the number of patterns of length n that can be stored in the SDM is independent of n. For large m, the number of patterns stored in the SDM increases linearly with m, and the capacity is not limited by n as it is for the Hopfield model.

Note that these expressions for the vector capacity were derived by setting the probability of getting an error in a particular bit (p_e) constant. The probability of getting M all of the pattern vectors correct is then given by $(1-p_e)^{nM}$.

Since n can be different for the two models, it is more relevant to compare the total number of bits (total information) stored in each model. For the Hopfield model the bit capacity scales as

$$bits = \frac{nn}{R^2}(1 - \frac{2}{R^2}), \tag{31}$$

whereas for the SDM, in the limit of large m, the number of stored bits goes as

$$bits = \frac{mn}{R^2}(1-\zeta^2/m). \tag{32}$$

Since the number of elements in T is n^2 and the number of elements in C is nm, the bit capacity for each model divided by the number of matrix elements is the same (in the limit of large n and m).[4] This shows that the asymptotic

[4] I have only divided the bit capacity for the SDM by the number of elements in C because these are the only elements that are variable and C contains the Hebbian

amount of information stored per matrix element in the SDM and the Hopfield model is the same, $\frac{1}{R^2}$.

Sequences

In an autoassociative memory, the system relaxes to one of the stored patterns and stays fixed in time (until a new input is presented). However, there are many situations where it is desirable to have the recalled patterns change sequentially in time. For example, a song can be remembered as a string of notes played in the correct sequence; cyclic patterns of muscle contractions are essential for walking, riding a bicycle, or dribbling a basketball.

Suppose that we wished to store a sequence of patterns in the SDM. Let the pattern vectors be given by $(\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^M)$. This sequence of patterns could be stored by having each pattern point to the next pattern in the sequence. Thus, for the SDM, the patterns would be stored as input-output pairs $(\mathbf{a}^\alpha, \mathbf{d}^\alpha)$, where $\mathbf{a}^\alpha = \mathbf{p}^\alpha$ and $\mathbf{d}^\alpha = \mathbf{p}^{\alpha+1}$ for $\alpha = 1,2,3,\ldots,M-1$. Convergence to this sequence works as follows: If the SDM is presented with an address that is close to $\mathbf{p}^1$ the read data will be close to $\mathbf{p}^2$. Iterating the system with $\mathbf{p}^2$ as the new input address, the read data will be even closer to $\mathbf{p}^3$. As this iterative process continues, the read data will converge to the stored sequence, with the next pattern in the sequence being presented at each time step.

The convergence statistics are essentially the same for sequential patterns as shown above for autoassociative patterns. Presented with $\mathbf{p}^\alpha$ as an input address, the signal for the stored sequence is found as before

$$< \text{signal} > = \delta m \, \mathbf{p}^{\alpha+1}. \tag{33}$$

learning coefficients. One could argue that the correct comparison should be the total number of elements in A and C, which would yield a factor of ½ for the SDM.

Thus, given $\mathbf{p}^{\alpha}$, the read data is expected to be $\mathbf{p}^{\alpha+1}$. Assuming that the patterns in the sequence are randomly chosen, the mean value of the noise is zero, with variance

$$< \sigma^2 > = (M-1)\delta^2 m\,(1+\delta^2(m-1)). \qquad (34)$$

Hence, the length of a sequence that can be stored in the SDM increases linearly with m for large m.

Attempting to store sequences like this in the Hopfield model is not very successful. If the length of the sequence is greater than 2, the values of the units at a given time typically become evenly distributed among all of the patterns. The reason for this failure is that the units are being updated asynchronously. Suppose that a sequence of patterns $(\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^M)$ is stored in the Hopfield network using $T = \sum_{\alpha=1}^{M-1} \mathbf{p}^{\alpha+1} \times \mathbf{p}^{\alpha}$. The state of each unit is updated according to the states of all the other units, so that if the network is presented with $\mathbf{p}^1$, the first few units will be updated and change their value to $\mathbf{p}^2$. After about $n/2$ units have changed their value, the local field for the other units now points half to $\mathbf{p}^2$ and half to $\mathbf{p}^3$. Thus, some of the units get updated to the states for $\mathbf{p}^3$ before the others achieve the correct values for $\mathbf{p}^2$. The net result after $\simeq Mn$ updates is that the units are typically distributed over all of the patterns in the sequence.

This failure to recall sequences is only an artifact of asynchronous updating. If the Hopfield model is modified to update the units synchronously, the Hopfield model will recall sequences just as described for the SDM. The number of patterns that can be stored in the sequence is determined by the signal-to-noise ratio and is equivalent to the capacity. Again, the length of the longest sequence stored in the Hopfield model is limited to a fraction of pattern size n, whereas in the SDM it is limited by the number of locations m, which can be varied independently of n.

Another method for storing sequences in Hopfield-like networks has been proposed independently by Kleinfeld (1986) and Sompolinsky and Kanter (1986) (see also Grossberg, 1971). These models relieve the problem created by asynchronous updating by using. a time-delayed sequential term The equations for updating are as follows:

$$u_i(t+1)=g\left(\sum_{j=1}^{n} T_{ij}u_j(t) + \sum_{j=1}^{n} D_{ij}u_j(t-k)\right), \qquad (35)$$

where $T = \mathbf{p}^{\alpha}\times\mathbf{p}^{\alpha}$, with $T_{ii} = 0$, $D = \mathbf{p}^{\alpha+1}\times\mathbf{p}^{\alpha}$, k is a delay of a few time steps, and the mean rate of asynchronous updating is n updates per unit time step. When presented a pattern close to $\mathbf{p}^1$, this system relaxes exactly to $\mathbf{p}^1$ within k time steps. Then in the next k time steps the units update to $\mathbf{p}^2$. Continuing in this fashion, the system will recover the stored sequence with a new pattern being presented every k time steps.

This time-delay storage algorithm has different dynamics than the synchronous SDM model. In the time-delay algorithm, the system allows time for the units to relax to the first pattern before proceeding on to the next pattern, whereas in the synchronous algorithms, the sequence is recalled imprecisely from imprecise input for the first few iterations and then correctly after that. In other words, convergence to the sequence takes place "on the fly" in the synchronous models — the system does not wait to zero in on the first pattern before proceeding on to recover the following patterns. This allows the synchronous algorithms to proceed k times as fast as the asynchronous time-delay algorithms with half as many (variable) matrix elements.

Time Delays and Hysteresis

In Kanerva's original work, he included the concept of time delays as a general way of storing sequences with hysteresis. The problem addressed by this is the following: Suppose we wish to store two sequences of patterns that

overlap. For example, the two pattern sequences (a,b,c,d,e,f,...) and (x,y,z,d,w,v,...) overlap at the pattern d. If the system only has knowledge of the present state, then when given the input d, it cannot decide whether to output w or e (see Figure 5). To store two such sequences, the system must have

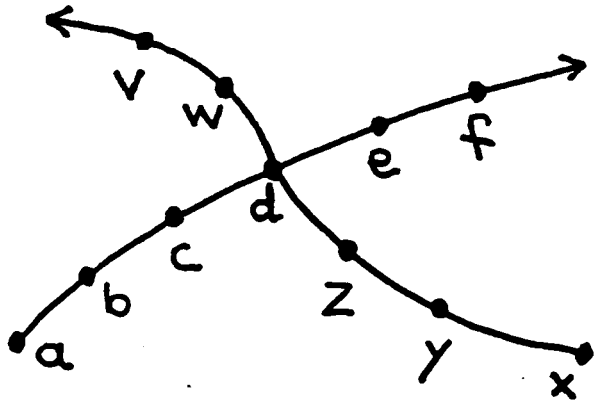

Figure 5. Two sequences are stored in the system. The first sequence (a,b,c,d,e,f,...) overlaps the second sequence (x,y,z,d,w,v,...) at d. The system must have some knowledge of the past states if it is to recover the sequences properly. This phenomenon can be thought of as momentum or hysteresis in a dynamical system.

some knowledge of the immediate past. Kanerva incorporates this idea into the SDM by using "folds." A system with $F+1$ folds has a time history of F past states. These F states may be over the past F time steps or they may go even further back in time, skipping some time steps. The algorithm for reading from the SDM with folds becomes

$$\mathbf{d}(t+1) = \mathbf{g}(C^0 \cdot \mathbf{s}(t) + C^1 \cdot \mathbf{s}(t-\tau_1) + C^F \cdot \mathbf{s}(t-\tau_F)), \tag{36}$$

where $\mathbf{s}(t-\tau_\gamma) = \theta_D(A\,\mathbf{d}(t-\tau_\gamma))$. To store the Q pattern sequences $(\mathbf{p}_1^1, \mathbf{p}_1^2, \ldots, \mathbf{p}_1^{M_1})$, $(\mathbf{p}_2^1, \mathbf{p}_2^2, \ldots, \mathbf{p}_2^{M_2})$,... $(\mathbf{p}_Q^1, \mathbf{p}_Q^2, \ldots, \mathbf{p}_Q^{M_Q})$, construct the matrix of the γ^{th} fold as follows:

$$C^\gamma = w_\gamma \sum_{\alpha=1}^{M_\gamma} \sum_{\beta=1}^{Q} \mathbf{p}_\beta^{\alpha+1} \times \mathbf{s}_\beta^{\alpha-\tau_\gamma}, \tag{37}$$

where any vector with a superscript less than 1 is taken to be zero,

$s_\beta^{\alpha-\tau_s} = \theta_D(A\, p_\beta^{\alpha-\tau_s})$, and w_γ is a weighting factor that would normally decrease with increasing γ.

Why do these folds work? Suppose that the system is presented with the pattern sequence $(p_1^1, p_1^2, \ldots, p_1^{M_1})$, with each pattern presented sequentially as input until the τ_F time step. For simplicity, assume that $w_\gamma = 1$ for all γ. Each term in Equation (36) will contribute a signal similar to the signal for the single-fold system. Thus, on this time step, the signal term coming from Equation (36) is

$$< \mathbf{signal}(t+1) > \; = F \delta m \, p_1^{\alpha+1}. \tag{38}$$

The signal term will have this value until the end of the pattern sequence is reached. The mean of the noise terms is zero, with variance

$$< \mathbf{noise}^2 > \; = F(M-1)\delta^2 m\,(1+\delta^2(m-1)). \tag{39}$$

Hence, the signal-to-noise ratio is $\sqrt{F}$ times as strong as it is for the SDM without folds.

Suppose further that the second stored pattern sequence happens to match the first stored sequence at $t = \eta$. The signal term would then be

$$\mathbf{signal}(t+1) = F \delta m \, p_1^{\eta+1} + \delta m \, p_2^{\eta+1}. \tag{40}$$

With no history of the past, $F = 1$, the signal is split between $p_1^{\eta+1}$ and $p_2^{\eta+1}$, and the output is ambiguous. However, for $F > 1$, the signal for the first pattern sequence dominates and allows retrieval of the remainder of the correct sequence.

Kanerva's formulation allows context to aid in the retrieval of stored sequences. Obviously, the Hopfield model could be extended to perform the same sort of function, but it would still be limited in the length of the sequences stored to some fraction of the word size times $\sqrt{F}$.

There are a large number of applications where context is an important consideration. For example, context is essential in speech recognition: If I say "bat," you don't know if I am talking about something used to hit a ball or a small winged mammal that flies at night, but if I say "the boy swung the bat," the ambiguity is relieved.

Time delays are prevalent in biological systems and seem to play an important role there also. For example, the signal time between two units depends on the length of the axon connecting the two units and the diameter of the axon. The above formulation might be useful in understanding the importance of these time delays.

Continuous Equations and Optimization Problems

The Hopfield model has also been formulated in terms of continuous dynamical equations (Hopfield, 1984). Continuous equations can also be written down for the SDM,

$$\frac{dv_i}{dt} = \frac{-v_i}{\tau} + \sum_{j=1}^{m} C_{ij}(\theta_D A_{ij} g(v_j)), \tag{41}$$

where τ is the characteristic self-relaxation time for the units, C is the same as in Equation (19), the state of the i^{th} input-output unit is $g(v_i)$, and θ_D and g are continuous analogs of the discrete functions described earlier. This continuous system has similar behavior to the discrete version of the autoassociative SDM, except that storing sequences in this system fails for the same reasons as described for asynchronously updated systems. To recover the properties of synchronous updating, a delay term could be included (Kleinfeld, 1986; Sompolinsky & Kanter, 1986) or some sort of explicit clocking term could be used.

Hopfield and Tank (1985) have used the continuous version of the

Hopfield (1984) model to gain approximate solutions to optimization problems. This is done by constructing a T matrix that yields an energy surface with minima at near-optimum solutions to the problem (see also Amari, 1971, 1974 and Cohen & Grossberg, 1983). So far, there has been no corresponding construction of an energy function for the SDM. Indeed, one would expect that such an energy function does not exist because of the asymmetric nature of C in the SDM and the fact that fixed points are not always reached under iteration of the autoassociative SDM.[5] Hence, performing optimization problems such as the traveling salesman problem with the SDM is not as straight-forward as it is with the Hopfield model. Nevertheless, there is a measure of the goodness of a read operation for the SDM. The standard deviation of the components of the field, **h**, achieves local maxima at the stored patterns in both autoassociative and heteroassociative memory. The standard deviation is not an exact analogy to the energy function of the Hopfield model, but this function might be useful for future work in optimization problems with the SDM.

Higher Order Interactions

Another generalization of the Hopfield model is to allow higher order nonlinear interactions between the units in the dynamical system. In the standard Hopfield model and the single-fold SDM model, the interactions are between pairs of units. This pairwise interaction leads to the linear-threshold rules given in Equations (2) and (14). However, if higher order interactions between the units are allowed, the threshold sum rule becomes a nonlinear tensor product. Lee *et al.* (1985) and Baldi and Venkatesh (1986) have shown that using higher order interactions can improve the capacity of the Hopfield model as well as some of its dynamical properties. I briefly describe the discrete

[5] This could just be an artifact of the synchronous update; fixed points might always be reached using asynchronous update.

formulation of these higher order systems below and investigate the information stored in these systems.

Consider n threshold units with interactions between $c+1$ other units. The updating algorithm for the system then becomes

$$u_i \leftarrow g\Big(\sum_{j_1 j_2 \cdots j_c}^{n} T_{ij_1j_2\cdots j_c} u_{j_1} u_{j_2} \cdots u_{j_c}\Big)\,, \tag{42}$$

where g is the same two-state threshold function as before, and u_i is the state of the i^{th} unit. To store the patterns $\mathbf{p}^1, \mathbf{p}^2, \cdots, \mathbf{p}^M$ in an autoassociative network, construct the $c+1$ order tensor T as

$$T_{ij_1j_2\cdots j_c} = \sum_{\alpha=1}^{M} p_i^{\alpha} p_{j_1}^{\alpha} p_{j_1}^{\alpha} \cdots p_{j_1}^{\alpha}. \tag{43}$$

The signal-to-noise ratio for this system is found in a straightforward manner similar to the calculation for the Hopfield model. The fidelity for this system is given by

$$R = \frac{n^{c/2}}{\sqrt{M-1}}, \tag{44}$$

and the number of bits stored in the system for large n is just

$$bits = \frac{n^{c+1}}{R^2}. \tag{45}$$

A similar generalization can be made for the SDM. The above system has generalized the Hebbian learning rule to include interactions between many units. If we generalize in the same manner for the SDM, the dynamical equations become

$$d_i \leftarrow g\Big(\sum_{j_1 j_2 \cdots j_c}^{m} C_{ij_1j_2\cdots j_c} s_{j_1} s_{j_2} \cdots s_{j_c}\Big)\,, \tag{46}$$

where the updating is done synchronously and $s_i = \theta_D\big(\sum_{j=1}^{n} A_{ij} a_j\big)$. Again, to

store the M patterns in an autoassociative network, construct the $c+1$ order tensor C

$$C_{ij_1j_2\cdots j_c} = \sum_{\alpha=1}^{M} p_i^{\alpha} s_{j_1}^{\alpha} s_{j_1}^{\alpha} \cdots s_{j_1}^{\alpha}. \tag{47}$$

where $s_i^{\alpha} = \theta_D(\sum_{j=1}^{n} A_{ij} p_j^{\alpha})$. The signal-to-noise ratio for this rule is

$$R = \frac{m^{c/2}}{\sqrt{(M-1)}(1+\frac{\zeta^2}{m})^{c/2}}. \tag{48}$$

From this expression the bit capacity for the SDM is found to be

$$bits = \frac{m^c n}{R^2} \tag{49}$$

in the limit of large m. The number of matrix elements in the model of Lee *et al.* is just n^{c+1}, whereas the number of elements for the SDM is nm^c. Hence, the number of bits per matrix element is the same for these two systems as well. Indeed, the number of bits per matrix element turns out to be identical to that for the linear systems discussed above ($\frac{1}{R^2}$). The indication is that this might be a general result for all Hebbian-type learning rules, and that the total information stored in systems of this type is a constant times the number of matrix elements, independent of the particular model.

Correlated Input Patterns

In the above associative memories, all of the patterns were taken to be randomly chosen, uniformly distributed binary vectors of length n. However, there are many applications where the set of input patterns is not uniformly distributed; the input patterns are correlated. For example, different human faces have many correlated features, but we are able to recognize a very large number of faces. In mathematical terms, the set κ of input patterns would not

be uniformly distributed over the entire space of 2^n possible patterns (see Figure 6). Let the probability distribution function for the Hamming distance

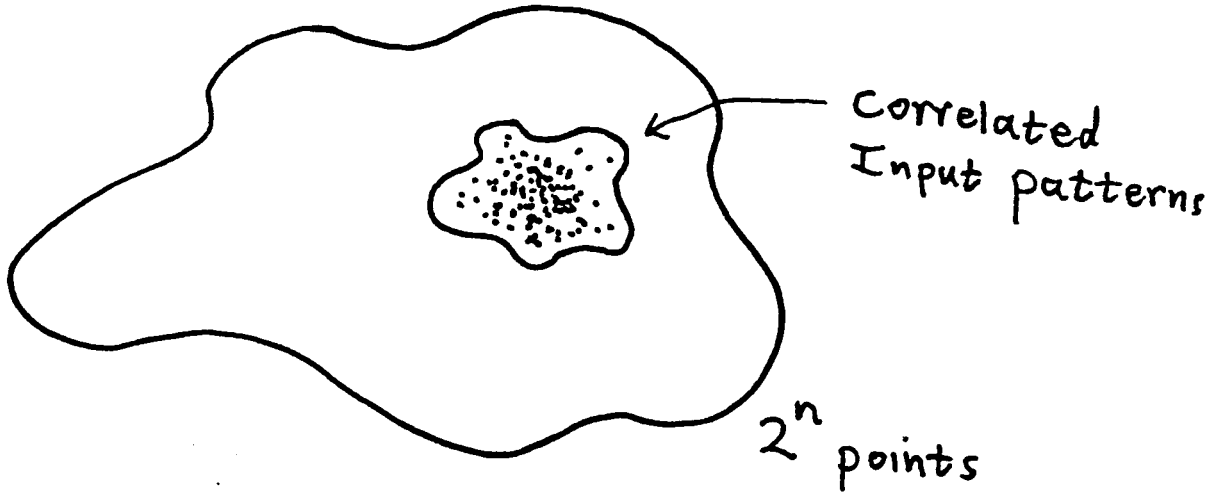

Figure 6. A schematic diagram of how the distribution of input patterns could be non-uniform in the space of all 2^n such patterns. The set κ could be distributed only over a very small region of the entire space. To recall the correlated patterns in this distribution, construct the rows of the matrix A with the same distribution and adjust the Hamming-distance threshold D so that the proper number of locations are selected.

between two randomly chosen vectors $\mathbf{p}^\alpha$ and $\mathbf{p}^\beta$ from the distribution κ be given by the function $\rho(d(\mathbf{p}^\alpha - \mathbf{p}^\beta))$. For the moment, assume that this distribution function is approximately Gaussian with mean γn, where $1/n < \gamma \leq n/2$. Recall that for a uniform distribution of patterns, $\rho(x)$ is just a binomial distribution with mean $n/2$.

The SDM can be generalized from Kanerva's original formulation so that correlated input patterns can be associated with output patterns. For the moment, assume that the distribution set κ and the probability density function $\rho(x)$ are known *a priori*. Instead of constructing the rows of the matrix A from the entire space of 2^n patterns, construct the rows of A from the distribution κ. Adjust the Hamming distance D so that $\zeta = \delta m$ locations are selected. In other words, adjust D so that the value of δ is the same as given above, where δ is determined by

$$\delta = \frac{\int_0^D \rho(x)\,dx}{\int_0^\infty \rho(x)\,dx}. \qquad (50)$$

Using the same distribution for the rows of A as the distribution of the patterns in κ, and using (50) to specify the choice of D, all of the above analysis is applicable.[6] The SDM will be able to recover data stored with correlated inputs with a fidelity given by Equation (28).

In general, the distribution κ could be much more complicated than described above. The set of patterns could be clustered in many correlated groups distributed over a large portion of the entire pattern space. In that case, the probability density function $\rho(\mathbf{p}^\alpha - \mathbf{p}^\beta)$ would depend on the patterns $\mathbf{p}$. These patterns could also be stored using the same distribution for the rows of A as the distribution of κ. In this case, the Hamming distance D would have to be determined dynamically to keep δ constant. This can be achieved by a feedback loop that compares the present value of δ to the desired value and adjusts D accordingly. Circuitry for this is straightforward to construct (Keeler & Denning 1986), and a mechanism of this sort is found in the cerebellum (Marr, 1969; see the Appendix).

There are certain biological memory functions that seem to be preprogrammed and have an *a priori* knowledge of what to expect as input from the outside world. This knowledge is equivalent to knowing something about the distribution function κ. However, what if the distribution function κ is not known *a priori?* In that case, we would need an algorithm for developing an A matrix that mimics the distribution of κ, and the elements of A would be

[6] Assuming randomly chosen output patterns. If the outputs are also correlated, the mean of the noise is not 0. However, if the distribution of outputs is also known, the system can still be made to work by adjusting the final threshold.

modifiable. There are many ways to build A to mimic κ. One such way is to start with a random A matrix and modify the entries of δ randomly chosen rows of A at each step according to the statistics of the most recent input patterns. Another method is to use competitive learning (Grossberg, 1976 & Kohonen, 1984) to achieve the proper distribution of A (see Keeler, 1987 for details).

Conclusion

The SDM model is attractive for its versatility and expandability. The number of patterns that can be stored in the SDM is independent of the size of the patterns, and the SDM can be used as an autoassociative or heteroassociative memory. The SDM can also be used to store sequences and can even retrieve correct sequences from contextual information by using folds. By adjusting the distribution of the A matrix, the SDM can also be used to associate patterns with correlated inputs.

The Hopfield model is attractive both for its simplicity and for its computational ability at approximating solutions to optimization problems. Moreover, the above investigation shows that the Hopfield model can also be used as a heteroassociative memory and can store sequences if synchronous updating is used.

Since the bit capacity per matrix element of the two networks is the same, what are the advantages of using one model instead of the other? The advantages would depend on the particular application. The SDM allows a greater number of patterns of a given size to be stored, but one has to pay a price for this in terms of the A matrix calculation. There are some applications where this extra calculation would be worth the effort. For example, storing many correlated patterns or sequences of patterns would be much easier to do in the SDM than in the Hopfield model. On the other hand, there are some

applications where the Hopfield model would be the best choice. For instance, if speed is the main constraint instead of capacity, it would be better to use the Hopfield model.

One of the main objections of using the Hopfield model as a model of biological neural networks is that the connections in the Hopfield model are symmetric. The above analysis demonstrates a way to analyze networks without the requirement of symmetric matrices. The SDM has no symmetry requirement and may therefore present a more realistic model of biological systems. Indeed, the SDM model is equivalent to Marr's (1969) model of the cerebellum (see Appendix). Marr's model was built up from the cellular level and includes a function for every neuron type found in the cerebellum. In that sense, the SDM is a very plausible model of biological memory.

Perhaps the most important feature of the above analysis is the similarity of the two models. These two memory models were developed from totally different points of view, yet they share many common features, and it was shown how these model can perform many of the same tasks. It was also shown that the bit capacity per matrix element is the same for the Hopfield model, the SDM, and the models with higher order interactions. These results indicate that there might be some universal behavior governing systems with Hebbian learning rules. There is still much work to be done to make the connection between these models and any biological system, but the similarities between these systems indicate that the SDM and the Hopfield model may have captured some of the essential underlying properties of neural networks. Future investigations and extensions of these models should prove fruitful for understanding biological systems and designing machines to mimic various biological tasks.

Acknowledgements

I thank Pentti Kanerva for useful discussions on this material and careful proofing of the text. I also thank David Rumelhart for useful comments on the text. Part of this work was done while visiting at the Research Institute for Advanced Computer Science at NASA Ames Research Center. This visit was made possible by a consortium agreement with Gary Chapman at NASA-Ames and Henry Abarbanel of the Institute for Nonlinear Science at U.C. San Diego.

Appendix: Relation of the SDM to the Cerebellum

The cerebellum is a part of the brain that is important in the coordination of complex muscle movements.[7] The neural organization of the cerebellum is highly regular: billions of Purkinje cells are stacked in parallel planes with about a hundred thousand axons from granule cells piercing these planes. The input to the cerebellum is through the mossy fibers which synapse on the granule cells (see Figure A1). The cerebellum also receives input from the inferior olive by means of the climbing fibers, which are in a one-to-one correspondence with the Purkinje cells and wrap themselves around the Purkinje cells. The sole output of the cerebellum is through the axons of the Purkinje cells.

David Marr (1969) modeled the cerebellum (see also Gilbert, 1974) in a fashion that is mathematically equivalent to the SDM The correspondence between the neurons in the cerebellum and the SDM model is as follows: The mossy fibers are the input to the SDM. The granule-cell axons are the select lines (this makes sense since the granule cells are the most populous neurons in the brain). It turns out that only a small fraction of the granule cells are firing at any one time. This fraction is the δ described above. If the granule cells receive the proper input from the mossy fibers, the granule cells fire, and they synapse on the Purkinje cells. The Purkinje cells fire if they receive enough input from the granule cells. Hence the Purkinje cells form the output layer, and the connections C_{ij} are the synapses between the Purkinje cells and the granule cells.

The hypothesis of Hebbian learning works as follows: The climbing fibers relay information that is to be stored in the Purkinje cells, and this information is written at the synapses that have active granule-cell input. This is the part of

[7] Most of the similarities in this section were pointed out by Kanerva (1984). I expand on his and Marr's (1969) description here (see also Gilbert, 1974).

the theory that is the most controversial and is the hardest to check experimentally.

There are three other types of neurons in the cerebellum that I have not mentioned so far. The first two are called basket cells and stellate cells, both of which synapse on the Purkinje cells. The function of the stellate cells is to adjust the threshold of the Purkinje cells, which would correspond to adjusting the threshold of the function g. The basket cells might not only adjust the threshold of the Purkinje cells, but they could also adjust the gain of the cells as well. The other type of cell is called a Golgi cell. The population of these cells is about 10% of the population of Purkinje cells. The Golgi cells receive input from the parallel fibers, and their axons synapse on the granule-cell-mossy-fiber clusters. The presumed function of the Golgi cell is to act as a feedback mechanism to keep the number of firing granule cells constant. This is analogous to a feedback mechanism for regulating δ described above.

Marr and Kanerva assumed that the synapses between the mossy fibers and the granule cells are fixed and that the inputs from the mossy fibers are random. It is apparent from the above discussion that there is no need for this assumption. These synapses might be fixed with genetically coded *a priori* knowledge of the expected inputs, or they could adjust in time to conform to the distribution of the mossy fiber input. This would allow differentiation between correlated inputs from the mossy fibers.

It is interesting that everything in the theory of the SDM fits in place in the model of the cerebellum even though the cerebellar cortex was not the original motivation for the SDM. The function of all of the cells in the cerebellum is not fully understood, and the mechanism for learning might be different than described above. However, the model seems to be faithful to everything that is known so far about the working of the cerebellum.

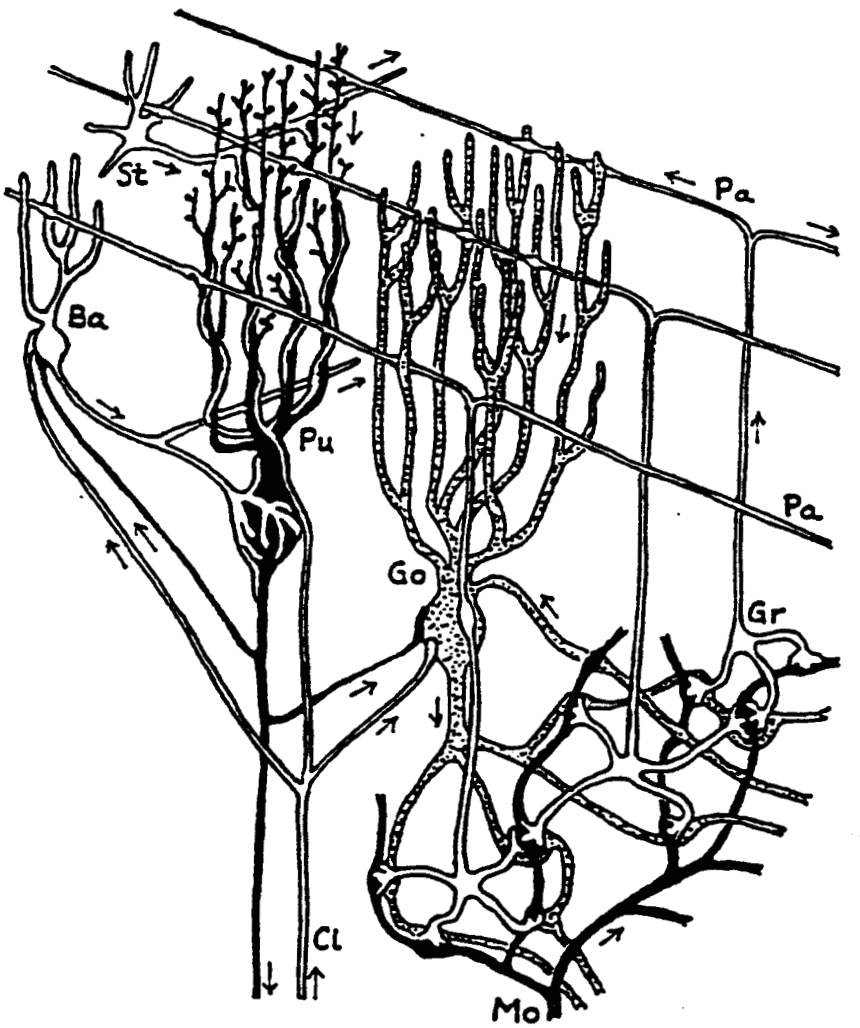

Figure A1. A sketch of the neurons of the cerebellar cortex. Pu= Purkinje cell (black), Go= Golgi cell (dotted), Gr= granule cell, Pa= parallel fiber, St= stellate cell, Ba= basket cell, Cl= climbing fiber, Mo= Mossy Fiber (black). Only a few of the cells are shown. Usually, there are about 100,000 parallel fibers in the parallel fiber bundle, but only about 500 of these are active at any one time. From "The Cortex of the Cerebellum" by Rodolfo R. Llinas.

References

Albus, J.S. (1971) A theory of Cerebellar Functions, *Mathematical Biosci.* **10** 25-61.

Amari, S., (1971) Characteristics of Randomly Connected Threshold-Element Networks and Network Systems, *Proc. IEEE,* **59,** 35-47.

Amari, S., (1974) A method of statistical neurodynamics, *Kybernetik* **14,** 201.

Amit, D. J., Gutfreund, H. & Sompolinsky, H. (1985) Storing an infinite number of patterns in a spin-glass model of neural networks, *Phys. Rev. Lett.* **55,** 1530-1533.

Anderson, J. A., Solverstein, J. W., Ritz, S. A. & Jones, R. S. (1977) *Psych. Rev.,* **84,** 412-451.

Baldi, P., and Venkatesh, S. S., (1986) The number of stable points for spin glasses and neural networks of higher orders, submitted to *Phys. Rev. Lett.*

Cohn, D. Kanerva, P., & Keeler, J. D. (1986) unpublished.

Gilbert, P. F. C., (1974) A Theory of memory that explains the function and structure of the cerebellum, *Brain Research,* **70,** 1-18

Cohn, M. A., Grossberg, S. (1983), Absolute stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks, *IEEE Trans. On Systems Man and Cybernetics,* **SMC-13.**

Denning, P. (1986) *A View of Kanerva's Sparse Distributed Memory,* RIACS Technical Report 86.14.

Grossberg, S. (1971), Embedding Fields: Underlying philosophy, mathematics, and applications to psychology, physiology, and anatomy, *J. Cyber.,* **1,** 28-50.

Grossberg, S. (1976), Adaptic pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors. *Biological Cybernetics* **23,** 121-134.

Hebb, D. O. (1949) *The Organization of Behavior.* John Wiley, New York.

Hopfield, J. J. (1982) Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proc. Natl. Acad. Sci. USA* **79** 2554-2558.

Hopfield, J. J. (1984) Neurons with graded response have collective computational properties like those of two state neurons, *Proc. Natl. Acad. Sci. USA* **81** 3088-3092.

Hopfield, J. J. & Tank, D. W., (1985) Neural Computation of Decisions in Optimization Problems, *Biological Cybern.* **52** 1-25.

Kanerva, P. (1984) *Self-propagating Search: A Unified Theory of Memory,* Stanford University Ph.D. Thesis, and Bradford Books (MIT Press). In press (1987 est).

Kanerva, P. (1986) *Parallel Structures in Human and Computer Memory,* RIACS Technical Report TR-86.2.

Keeler, J. D. (1987), University of California, San Diego, Ph.D. Thesis, in preparation.

Keeler, J. D. & Denning, P. J., (1986) *Notes on implementation of Sparsely Distributed Memory,* RIACS Technical Report, 86.15.

Kirkpatrick, S. & Sherrington, D. (1978) *Phys Rev.* **17** 4384-4405.

Kleinfeld, D. (1986) Sequential State Generation by Model Neural Networks, submitted to PNAS

Kohonen, T. (1980) *Content Addressable Memories,* Springer-Verlag, Berlin.

Kohonen, T. (1984) *Self-organization and Associative Memory,* Springer-Verlag, Berlin.

Lee, Y. C.; Doolen, G.; Chen, H. H.; Sun, G. Z.; Maxwell, T.; Lee, H. Y.; & Giles, L. (1985) Machine learning using a higher order correlation network, *Physica D,* **23,** Conference on Evolution Games & Learning.

Little, W. A. & Shaw, G. L.(1978)*Math. Biosci.* **39,** 281-289.

Marr, D., (1969) *J. of Phisiol.*, A Theory of Cerebellar Cortex, **202,** 437-470.

McCulloch, W. S. & Pitts, W. (1943), A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.* **5,** 115-133.

McEliece, R. J., Posner, E. C., Rodemich, E. R., & Venkatesh, S. S. (1986), The Capacity of the Hopfield Associative Memory, *IEEE Trans. on Information Theory.*

Nakano, K. (1972), Association - A model of associative memory, *IEEE Trans. Sys. Man Cyber.* **2,** 380-387.

Shannon, C. E., (1948), A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, **27,** 379,623 (Reprinted in Shannon and Weaver 1949) .

Sompolinsky, H. & Kanter, I. (1986) Temporal Association in Asymmetric Neural Networks, submitted to *Phys. Rev. Letters.*

Willshaw, D. J., Buneman, O. P. & Longuet-Higgins, H. C., (1969) Non-holographic Associative Memory *Nature,* **222** 960-962.