Research Institute for Advanced Computer Science
NASA Ames Research Center

Sparse Distributed Memory and Related Models

Pentti Kanerva

RIACS Technical Report 92.10

April 1992

Sparse Distributed Memory and Related Models

Pentti Kanerva

The Research Institute for Advanced Computer Science is operated by
Universities Space Research Association (USRA),
The American City Building, Suite 311, Columbia, MD 21044, (301)730-2656.

Work reported herein was supported by the National Aeronautics and Space Administration (NASA) under Cooperative Agreement NC2-387 with the Universities Space Research Association (USRA).

RIACS Technical Report 92.10

April 1992

Sparse Distributed Memory and Related Models

Pentti Kanerva

Research Institute for Advanced Computer Science
Mail Stop T041-5
NASA Ames Research Center
Moffett Field, CA 94035
e-mail: kanerva@riacs.edu

Abstract

This paper describes sparse distributed memory (SDM) as a neural-net associative memory. It is characterized by two weight matrices and by a large internal dimension—the number of hidden units is much larger than the number of input or output units. The first matrix, **A**, is fixed and possibly random, and the second matrix, **C**, is modifiable. The paper compares and contrasts SDM to (1) computer memory, (2) correlation-matrix memory, (3) feed-forward artificial neural network, (4) cortex of the cerebellum, (5) Marr and Albus models of the cerebellum, and (6) Albus' cerebellar model arithmetic computer (CMAC). Several variations of the basic SDM design are discussed: the selected-coordinate and hyperplane designs of Jaeckel, the pseudorandom associative neural memory of Hassoun, and SDM with real-valued input variables by Prager and Fallside. SDM research conducted mainly at RIACS in 1986–1991 is highlighted.

To appear as Chapter 3 in H. M. Hassoun, ed. *Associative Neural Memories: Theory and Implementation*. New York: Oxford University Press.

Contents

1. Introduction

This chapter describes one basic model of associative memory, called the sparse distributed memory, and relates it to other models and circuits: to ordinary computer memory, to correlation-matrix memories, to feed-forward artificial neural nets, to neural circuits in the brain, and to associative-memory models of the cerebellum. Presenting the various designs within one framework will hopefully help the reader see the similarities and the differences in designs that are often described in different ways.

1.1 Sparse Distributed Memory as a Model of Human Long-Term Memory

Sparse Distributed Memory (SDM) was developed as a mathematical model of human long-term memory (Kanerva 1988). The pursuit of a simple idea led to the discovery of the model, namely, that the distances between concepts in our minds correspond to the distances between points of a high-dimensional space. In what follows, 'high-dimensional' means that the number of dimensions is at least in the hundreds, although smaller numbers of dimensions are often found in examples.

If a concept, or a percept, or a moment of experience, or a piece of information in memory—a point of interest—is represented by a high-dimensional (or "long") vector, the representation need not be exact. This follows from the distribution of points of a high-dimensional space: Any point of the space that might be a point of interest is relatively far from most of the space and from other points of interest. Therefore, a point of interest can be represented with considerable slop before it is confused with other points of interest. In this sense, long vectors are fault-tolerant or robust, and a device based on them can take advantage of the robustness.

This corresponds beautifully to how humans and animals with advanced sensory systems and brains work. The signals received by us at two different times are hardly ever identical, and yet we can identify the source of the signal as a *specific* individual, object, place, scene, thing. The representations used by the brain must allow for such identification, in fact, they must make the identification nearly automatic, and high-dimensional vectors as internal representations of things do that.

Another property of high-dimensional spaces also has to do with the distances between points. If we take two points (of interest) at random, they are relatively far from each other, on the average: they are uncorrelated. However, there are many points between the two that are close to both, in the sense that the amount of space around an intermediate point—in a hypersphere—that contains both of the two original points is very small. This corresponds to the relative ease with which we can find a concept that links two unrelated concepts.

Strictly speaking, a mathematical space need not be a high-dimensional vector

space to have the desired properties; it needs to be a huge space, with an appropriate similarity measure for pairs of points, but the measure need not define a metric on the space.

The important properties of high-dimensional spaces are evident even with the simplest of such spaces—that is, when the dimensions are binary. Therefore, the sparse distributed memory model was developed using long (i.e., high-dimensional) binary vectors or words. The memory is addressed by such words, and such words are stored and retrieved as data.

The following two examples demonstrate the memory's robustness in dealing with approximate data. The memory works with 256-bit words: it is addressed by them, and it stores and retrieves them. On top of Figure 2–1 are nine similar (20% noisy) 256-bit words. To help us compare long words, their 256 bits are laid on a 16-by-16 grid, with 1s shown in black. The noise-free prototype word was designed in the shape of a circle within the grid. (This example is confusing in that it might be taken to imply that humans recognize circles based on stored retinal images of circles. No such claim is intended.) The nine noisy words were stored in a sparse distributed memory autoassociatively, meaning that each word was stored with itself as the address. When a tenth noisy word (bottom left), different from the nine, was used as the address, a relatively noise-free 11th word was retrieved (bottom middle), and with that as the address, a nearly noise-free 12th word was retrieved (bottom right), which in turn retrieved itself. This example demonstrates the memory's tendency to construct a prototype from noisy data.

((**FIGURE** 2–1. Nine noisy words are stored ...))

Figure 2–2 demonstrates sequence storage and recall. Six words, shaped as Roman numerals, are stored in a linked list: I is used as the address to store II, II is used as the address to store III, and so forth. Any of the words I–V can then be used to recall the rest of the sequence. For example, III will retrieve IV will retrieve V will retrieve VI. The retrieval cue for the sequence can be noisy, as demonstrated at the bottom of the figure. As the retrieval progresses, a retrieved word, which then serves as the next address, is less and less noisy. This example resembles human ability to find a familiar tune by hearing a piece of it in the middle, and to recall the rest. This kind of recall applies to a multitude of human and animal skills.

((**FIGURE** 2–2. Recalling a stored sequence ...))

2. SDM as a Random-Access Memory

Except for the lengths of the address and data words, the memory resembles ordinary computer memory. It is a generalized random-access memory for long words, as will be seen shortly, and its construction and operation can be explained in terms of an ordinary random-access memory. We will start by describing an ordinary random-access memory.

2.1 Random-Access Memory

A random-access memory (RAM) is an array of M addressable storage registers or memory locations of fixed capacity. A location's place in the memory array is called the location's *address*, and the value stored in the register is called the location's *contents*. Figure 2–3 represents such a memory, and a horizontal row through the figure represents one memory location. The first location is shown shaded. The addresses of the locations are on the left, in matrix **A**, and the contents are on the right, in matrix **C**.

((**FIGURE** 2–3. Organization of a random-access memory.))

A memory with a million locations ($M = 2^{20}$) is addressed by 20-bit words. The length of the address will be denoted by N ($N = 20$ in Fig. 2–3). The capacity of a location is referred to as the memory's *word size*, U ($U = 32$ bits in Fig. 2–3), and the capacity of the entire memory is defined conventionally as the word size multiplied by the number of memory locations (i.e., $M \times U$ bits).

Storage and retrieval happen one word at a time through three special registers: the *address register*, for an N-bit address into the memory array; the *word-in register*, for a U-bit word that is to be stored in memory; and the *word-out register*, for a U-bit word retrieved from memory. To store the word **w** in location **x** (the location's address is used as a name for the location), **x** is placed in the address register, **w** is placed in the word-in register, and a write-into-memory command is issued. Consequently, **w** replaces the old contents of location **x**, while all other locations remain unchanged. To retrieve the word **w** that was last stored in location **x**, **x** is placed in the address register and a read-from-memory command is issued. The result **w** appears in the word-out register. The figure shows (a possible) state of the memory after **w** = 010...110 has been stored in location **x** = 000...011 (the word-in register holds **w**) and then retrieved from the same location (the address register holds **x**).

Between matrices **A** and **C** in the figure is an *activation vector*, **y**. Its components are 0s except for one 1, which indicates the memory location that is selected for reading or writing (i.e., the location's address matches the address register). In a hardware realization of a random-access memory, a location's activation is determined by an address-decoder circuit, so that the address matrix **A** is implicit. However, the contents matrix **C** is an explicit array of $2^{20} \times 32$ one-bit registers or flip-flops.

2.2 Sparse Distributed Memory

Figure 2–4 represents a sparse distributed memory. From the outside, it is like a random-access memory: it has the same three special registers—address, word-in, and word-out—and they are used in the same way when words are stored and

retrieved, except that these registers are large (e.g., $N = U = 1{,}000$).

((**FIGURE** 2–4. Organization of a sparse distributed memory.))

Construction. The internal organization of sparse distributed memory, likewise, is an array of addressable storage locations of fixed capacity. However, since the addresses are long, it is impossible to build a hardware location—a *hard location*, for short—for each of the 2^N addresses. (Neither is it necessary, considering the enormous capacity that such a memory would have.)

A memory of reasonable size and capacity can be built by taking a reasonably large sample of the 2^N addresses and by building a hard location for each address in the sample. Let M be the size of the sample: we want a memory with M locations ($M = 1{,}000{,}000$ in Fig. 2–4). The sample can be chosen in many ways, and only some will be considered here.

A good choice of addresses for the hard locations depends on the data to be stored in the memory. The data consist of the words to be stored and of the addresses used in storing them. For simplicity, we assume in the basic model that the data are distributed randomly and uniformly (i.e., bits are independent of each other, and 0s and 1s are equally likely, both in the words being stored and in the addresses used for storing them). Then the M hard locations can be picked at random; that is to say, we can take a uniform random sample, of size M, of all N-bit addresses. Such a choice of locations is shown in Figure 2–4, where the addresses of the locations are given in matrix **A** and the contents are given in matrix **C**, and where a row through the figure represents a hard location, just as in Figure 2–3 (row $\mathbf{A}_m$ of matrix **A** is the mth hard address, and $\mathbf{C}_m$ is the contents of location $\mathbf{A}_m$; as with RAM, we use $\mathbf{A}_m$ to name the mth location).

Activation. In a random-access memory, to store or retrieve a word with **x** as the address, **x** is placed in the (20-bit) address register, which activates location **x**. We say that the address register *points* to location **x**, and that whatever location the address register points to is activated. This does not work with a sparse distributed memory because its (1,000-bit) address register never—practically never—points to a hard location because the hard locations are so few compared to the number of possible addresses (e.g., 1,000,000 hard addresses *vs.* $2^{1{,}000}$ possible addresses; matrix **A** is an exceedingly sparse sampling of the address space).

To compensate for the extreme sparseness of the memory, a *set* of *nearby* locations is activated at once, for example, all the locations that are within a certain *distance* from **x**. Since the addresses are binary, we can use Hamming distance, which is the number of places at which two binary vectors differ. Thus, in a sparse distributed memory, the mth location is activated by **x** (which is in the address register) if the Hamming distance between **x** and the location's address $\mathbf{A}_m$ is below or equal to a threshold value H (H stands for a [Hamming] radius of activation). The threshold is chosen so that but a small fraction of the hard locations are activated by

any given **x**. When the hard addresses **A** are a uniform random sample of the N-dimensional address space, the binomial distribution with parameters N and 1/2 can be used to find the activation radius H that corresponds to a given probability p of activating a location. Notice that, in a random-access memory, a location is activated only if its address matches **x**, meaning that $H = 0$.

Vectors **d** and **y** in Figure 2–4 show the activation of locations by address **x**. The distance vector **d** gives the Hamming distances from the address register to each of the hard locations, and the 1s of the activation vector **y** mark the locations that are close enough to **x** to be activated by it: $\mathbf{y}_m = 1$ if $\mathbf{d}_m \leq H$, and $\mathbf{y}_m = 0$ otherwise, where $\mathbf{d}_m = h(\mathbf{x}, \mathbf{A}_m)$ is the Hamming distance from **x** to location $\mathbf{A}_m$. The number of 1s in **y** therefore equals the size of the set activated by **x**.

Figure 2–5 is another way of representing the activation of locations. The large circle represents the space of 2^N addresses. Each tiny square is a hard location, and its position within the large circle represents the location's addresses. The small circle around **x** includes the locations that are within H bits of **x** and that therefore are activated by **x**.

((**FIGURE** 2–5. Address space, hard locations, and the set ...))

Storage. To store U-bit words, a hard location has U up–down counters. The range of a counter can be small, for example, the integers from −15 to 15. The U counters for each of the M hard locations constitute the $M \times U$ contents matrix, **C**, shown on the right in Figure 2–4, and they correspond to the $M \times U$ flip-flops of Figure 2–3. We will assume that all counters are initially set to zero.

When **x** is used as the storage address for the word **w**, **w** is stored in each of the locations activated by **x**. Thus, multiple copies of **w** are stored; in other words, **w** is distributed over a (small) number of locations. The word **w** is stored in, or written into, an active location as follows: Each 1-bit of **w** increments, and each 0-bit of **w** decrements, the corresponding counter of the location. This is equivalent to saying that the word $\mathbf{w}'$ of −1s and 1s is added (vector addition) to the contents of each active location, where $\mathbf{w}'$ is gotten from **w** by replacing 0s with −1s. Furthermore, the counters in **C** are not incremented or decremented past their limits (i.e., overflow and underflow are lost).

Figure 2–4 depicts the memory after the word **w** = 010...110 (in the word-in register) has been stored with **x** = 100...101 as the address (in the address register). Several locations are shown as selected, and the vector $\mathbf{w}' = (-1, 1, -1, \ldots, 1, 1, -1)$ has been added to their contents. The figure also shows that many locations have been selected for writing in the past (e.g., the first location has nonzero counters), that the last location appears never to have been selected, and that **w** appears to be the first word written into the selected location near the bottom of the memory (the location contains $\mathbf{w}'$). Notice that a positive value of a counter, +5, say, tells that five more 1s than 0s have been stored in it; similarly, −5 tells that five more 0s than 1s

have been stored (provided that the capacity of the counter has never been exceeded).

Retrieval. When **x** is used as the retrieval address, the locations activated by **x** are pooled as follows: their contents are accumulated (vector addition) into a vector of U sums, **s**, and the sums are compared to a threshold value 0 to get an output vector **z**, which then appears in the word-out register ($\mathbf{z}_u = 1$ iff $\mathbf{s}_u > 0$; **s** and **z** are below matrix **C** in Fig. 2–4). This pooling constitutes a majority rule, in the sense that the uth output bit is 1 if, and only if, more 1s than 0s have been stored in the uth counters of the activated locations; otherwise, the output bit is 0.

In Figure 2–4 the word retrieved, **z**, is the same as, or very similar to, the word **w** that was stored, for the following reason: The same **x** is used as both storage and retrieval address, so that the same set of locations is activated both times. In storing, each active location receives one copy of **w**′, as described above; in retrieving, we get back *all* of them, plus a few copies of many other words that have been stored. This biases the sums, **s**, in the direction of **w**′, so that **w** is a likely result after thresholding. This principle holds even when the retrieval address is not exactly **x** but is close to it. Then we get back *most* of the copies of **w**′.

The ideas of storing multiple copies of target words in memory, and of retrieving the most likely target word based on the majority rule, are found already in the *redundant hash addressing* method of Kohonen and Reuhkala (1978, Kohonen 1980). The method of realizing these ideas in redundant hash addressing is very different from their realization in a sparse distributed memory.

Retrieval and memory capacity will be analyzed statistically at the end of the next section, after a uniform set of symbols and conventions for the remainder of this chapter has been established. We will note here, however, that the intersections of activation sets play a key role in the analysis, for they appear as weights for the words stored in the memory when the sum vector **s** is evaluated.

Random-Access Memory as a Special Case. One more comment about a random-access memory: Proper choice of parameters for a sparse distributed memory yields an ordinary random-access memory. First, the address matrix **A** must contain all 2^N addresses; second, the activation radius H must be zero; and, third, the capacity of each counter in **C** must be one bit. The first condition guarantees that every possible address **x** points to at least one hard location. The second condition guarantees that only a location that is pointed to is activated. The third condition guarantees that when a word is written into a location, it replaces the location's old contents, because overflow and underflow are lost. In memory retrieval, the contents of all active locations are added together; in this case, the sum is over one or more locations with hard address **x**. Any particular coordinate of the sum is zero if the word last written (with address **x**) has a 0 in that position; and it is positive if the word has a 1, so that after thresholding we get the word last written

with address **x**. Therefore, the sparse distributed memory is a generalization of the random-access memory.

Parallel Realization. Storing a word, or retrieving a word, in a sparse distributed memory involves massive computation. The contents of the address register are compared to each hard address, to determine which locations to activate. For the model memory with a million locations, this means computing one-million Hamming distances involving 1,000 bits each, and comparing the distances to a threshold. This is very time-consuming if done serially. However, the activations of the hard locations are independent of each other so that they can be computed in parallel; once the address is broadcast to all the locations, million-fold parallelism is possible. The addressing computation that determines the set of active locations corresponds to address decoding by the address-decoder circuit in a random-access memory.

In storing a word, each column of counters in matrix **C** (see Fig. 2–4) can be updated independently of all other columns, so that there is an opportunity for thousand-fold parallelism when 1,000-bit words are stored. Similarly, in retrieving a 1,000-bit word, there is an opportunity for thousand-fold parallelism. Further parallelism is achieved by updating many locations at once when a word is stored, and by accumulating many partial sums at once when a word is retrieved. It appears that neural circuits in the brain are wired for these kinds of parallelism.

3. SDM as a Matrix Memory

The construction of the memory was described above in terms of vectors and matrices. We will now see that its operation is described naturally in vector–matrix notation. Such description is convenient in relating the sparse distributed memory to the correlation-matrix memories described by Anderson (1968) and Kohonen (1972)—see also Hopfield (1982), Kohonen (1984), Willshaw (1981), and Chapter 1 by Hassoun—and in relating it to many other kinds of artificial neural networks. The notation will also be used for describing variations and generalizations of the basic sparse distributed memory model.

3.1 Notation

In comparing the memory to a random-access memory, it is convenient to express binary addresses and words in 0s and 1s. In comparing it to a matrix memory, however, it is more convenient to express them in −1s and 1s (also called *bipolar* representation). This transformation is already implicit in the storage algorithm described above: a binary word **w** of 0s and 1s is stored by adding the corresponding word **w′** of −1s and 1s into (the contents of) the active locations. From here on, we assume that the binary components of **A** and **x** (and of **w** and **z**) are −1s and 1s, and whether *bit* refers to 0 and 1 or to −1 and 1 will depend on the context.

How is the activation of a location determined after this transformation? In the same way as before, provided that Hamming distance is defined as the number of places at which two vectors differ. However, we can also use the inner product (scalar product, dot product) of the hard address $\mathbf{A}_m$ and the address register $\mathbf{x}$ to measure their similarity: $d = d(\mathbf{A}_m, \mathbf{x}) = \mathbf{A}_m \cdot \mathbf{x}$. It ranges from $-N$ to N ($d = N$ means that the two addresses are most similar—they are identical), and it relates linearly to the Hamming distance, which ranges from 0 to N (0 means identical). Therefore, Hamming distance $h(\mathbf{A}_m, \mathbf{x}) \le H$ if, and only if, $\mathbf{A}_m \cdot \mathbf{x} \ge N - 2H\ (= D)$. In a computer simulation of the memory, however, the exclusive-or (XOR) operation on addresses of 0s and 1s usually results in the most efficient computation of distances and of the activation vector $\mathbf{y}$.

The following typographic conventions will be used:

s	italic lowercase for a scalar or a function name.
S	italic uppercase for a scalar upper bound or a threshold.
$\mathbf{v}$	bold lowercase for a (column) vector.
$\mathbf{v}_i$	ith component of a vector, a scalar.
$\mathbf{M}$	bold uppercase for a matrix.
$\mathbf{M}_i$	ith row of a matrix, a (column) vector.
$\mathbf{M}_{\cdot,j}$	jth column of a matrix, a (column) vector.
$\mathbf{M}_{i,j}$	scalar component of a matrix.
$\mathbf{M}^{\mathrm{T}}$	transpose of a matrix (or of a vector).
$\cdot$	scalar (inner) product of two vectors: $\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^{\mathrm{T}}\mathbf{v}$.
$\square$	matrix (outer) product of two vectors: $\mathbf{u}\square\mathbf{v} = \mathbf{u}\mathbf{v}^{\mathrm{T}}$.
$n = 1, 2, 3, \ldots, N$	index into the bits of an address.
$u = 1, 2, 3, \ldots, U$	index into the bits of a word.
$t = 1, 2, 3, \ldots, T$	index into the data.
$m = 1, 2, 3, \ldots, M$	index into the hard locations.

3.2 Memory Parameters

The sparse distributed memory, as a matrix memory, is described below in terms of its parameters, progressing with the information flow from upper left to lower right in Figure 2–4. *Sample memory* refers to a memory whose parameter values appear in parentheses in the descriptions below, as in "(e.g., N = 1,000)".

The *external dimensions* of the memory are given by:

N Address length; dimension of the address space; input dimension (e.g., N = 1,000). Small demonstrations can be made with N as small as 25, but $N > 100$ is recommended, as the properties of high-dimensional spaces will then be evident.

U Word length; the number of bits (−1s and 1s) in the words stored; output dimension (e.g., U = 1,000). The minimum, $U = 1$, corresponds to classifying the data into two classes. If $U = N$, it is possible to store words autoassociatively and to

store sequences of words as pointer chains, as demonstrated in Figures 2–1 and 2–2.

The *data set* to be stored—the *training set* (**X**, **W**)—is given by:

T Training-set size; number of elements in the data set (e.g., $T = 10{,}000$).

X Data-address matrix; T training addresses; $T \times N$ −1s and 1s (e.g., uniform random).

W Data-word matrix; T training words; $T \times U$ −1s and 1s (e.g., uniform random). Autoassociative data (self-addressing) means that $\mathbf{X} = \mathbf{W}$, and sequence data means that $\mathbf{X}_t = \mathbf{W}_{t-1}$ $(t > 1)$.

The memory's *internal parameters* are:

M Memory size; number of hard locations (e.g., $M = 1{,}000{,}000$). Memory needs to be sufficient for the data being stored and for the amount of noise to be tolerated in retrieval. Memory capacity is low, so that T should be 1–5 percent of M (T is the number of stored patterns; storing many noisy versions of the same pattern [cf. Fig. 2–1] counts as storing one pattern, or as storing few).

A Hard-address matrix; M hard addresses; $M \times N$ −1s and 1s (e.g., uniform random). This matrix is fixed. Efficient use of memory requires that **A** correspond to the set of data addresses **X** (see Sec. 8 on SDM research).

p Probability of activation (e.g., $p = 0.000445$; "ideally," $p = 0.000368$). This important parameter determines the number of hard locations that are activated, on the average, by an address, which, in turn, determines how well stored words are retrieved. The best p maximizes the signal (due to the target word that is being retrieved) relative to the noise (due to all other stored words) in the sum, **s**, and is approximately $(2MT)^{-1/3}$ (see end of this section, where signal, noise, and memory capacity are discussed).

H Radius of activation (e.g., $H = 447$ bits). The binomial distribution or its normal approximation can be used to find the (Hamming) radius for a given probability. For the sample memory, optimal p is 0.000368, so that about 368 locations should be activated at a time. Radius $H = 446$ captures 354 locations, and $H = 447$ captures 445 locations, on the average. We choose the latter.

D Activation threshold on similarity (e.g., $D = 106$). This threshold is related to the radius of activation by $D = N - 2H$, so that $D = 108$ and $D = 106$ correspond to the two values of H given above.

c Range of a counter in the $M \times U$ contents matrix **C** (e.g., $c = \{-15, -14, -13, \ldots, 14, 15\}$). If the range is one bit ($c = \{0, 1\}$), the contents of a location are determined wholly by the most-recent word written into the location. An 8-bit byte, an integer variable, and a floating-point variable are convenient counters in computer simulations of the memory.

The following variables describe the *memory's state and operation*:

x Storage or retrieval address; contents of the address register; N −1s and 1s (e.g., $\mathbf{x} = \mathbf{X}_t$).

d Similarity vector; M integers in $\{-N, -N+2, -N+4, \ldots, N-2, N\}$. Since the similarity between the mth hard address and the address register is given by their inner product $\mathbf{A}_m \cdot \mathbf{x}$ (see Sec. 3.1 on Notation), the similarity vector can be expressed as $\mathbf{d} = \mathbf{Ax}$.

y Activation vector; M 0s and 1s. The similarity vector **d** is converted into the activation vector **y** by the (nonlinear) threshold function y defined by $y(\mathbf{d}) = \mathbf{y}$, where $\mathbf{y}_m = 1$ if $\mathbf{d}_m \geq D$, and $\mathbf{y}_m = 0$ otherwise. The number of 1s in **y**, $|\mathbf{y}|$, is small compared to the number of 0s ($|\mathbf{y}| \approx pM$); the activation vector is a very sparse vector in a very-high-dimensional space. Notice that this is the only vector of 0s and 1s; all other binary vectors consist of −1s and 1s.

w Input word; U −1s and 1s (e.g., $\mathbf{w} = \mathbf{W}_t$).

C Contents matrix; $U \times M$ up–down counters with range c, initial value usually assumed to be 0. Since the word **w** is stored in active location $\mathbf{A}_m$ (i.e., when $\mathbf{y}_m = 1$) by adding **w** into the location's contents $\mathbf{C}_m$, it is stored in *all* active locations indicated by **y** by adding the (outer-product) matrix $\mathbf{y}\square\mathbf{w}$ (most of whose rows are 0) into **C**, so that $\mathbf{C} := \mathbf{C} + \mathbf{y}\square\mathbf{w}$, where := means substitution, and where addition beyond the range of a counter is ignored. This is known as the outer-product, or Hebbian, learning rule.

s Sum vector; U sums (each sum has [at most] $|\mathbf{y}|$ nonzero terms). Because the sum vector is made up of the contents of the active locations, it can be expressed as $\mathbf{s} = \mathbf{C}^T\mathbf{y}$. The U sums give us the final output word **z**, but they also tell us how reliable each of the output bits is. The further a sum is from the threshold, the stronger is the memory's evidence for the corresponding output bit.

z Output word; U −1s and 1s. The sum vector **s** is converted into the output vector **z** by the (nonlinear) threshold function z defined by $z(\mathbf{s}) = \mathbf{z}$, where $\mathbf{z}_u = 1$ if $\mathbf{s}_u > 0$, and $\mathbf{z}_u = -1$ otherwise.

In summary, storing the word **w** into the memory with **x** as the address can be expressed as

$$\mathbf{C} := \mathbf{C} + y(\mathbf{Ax})\square\mathbf{w}$$

and retrieving the word **z** corresponding to the address **x** can be expressed as

$$\mathbf{z} = z(\mathbf{C}^T y(\mathbf{Ax}))$$

3.3 Summary Specification

The following matrices describe the memory's operation on the data set—the training set (**X**, **W**)—as a whole:

D $T \times M$ matrix of similarities corresponding to the data addresses **X**: $\mathbf{D} = (\mathbf{AX}^T)^T = \mathbf{XA}^T$.

Y Corresponding $T \times M$ matrix of activations: $\mathbf{Y} = y(\mathbf{D})$.

S $T \times U$ matrix of sums for the data set: $\mathbf{S} = \mathbf{YC}$.

Z Corresponding $T \times U$ matrix of output words: $\mathbf{Z} = z(\mathbf{S}) = z(\mathbf{YC})$.

If the initial contents of the memory are 0, and if the capacities of the counters are never exceeded, storing the T-element data set yields memory contents

$$\mathbf{C} = \sum_{t=1}^{T} \mathbf{Y}_t \square \mathbf{W}_t = \sum_{t=1}^{T} y(\mathbf{AX}_t) \square \mathbf{W}_t$$

This expression for **C** follows from the outer-product learning rule (see **C** above), as it is the sum of T matrices, each of which represents an item in the data set. However, **C** can be viewed equivalently as a matrix of $M \times U$ inner products $\mathbf{C}_{m,u}$ of pairs of vectors of length T. One set of these vectors is the M columns of **Y**, and the other set is the U columns of **W**, so that $\mathbf{C}_{m,u} = \mathbf{Y}_{\cdot,m} \cdot \mathbf{W}_{\cdot,u}$, and

$$\mathbf{C} = \mathbf{Y}^{\mathrm{T}}\mathbf{W} = y(\mathbf{AX}^{\mathrm{T}})\mathbf{W}$$

The accuracy of recall of the training set after it has been stored in memory, is then given by

$$\begin{aligned} \mathbf{Z} - \mathbf{W} &= z(\mathbf{YC}) - \mathbf{W} \\ &= z(\mathbf{YY}^{\mathrm{T}}\mathbf{W}) - \mathbf{W} \end{aligned}$$

This form is convenient in the mathematical analysis of the memory. For example, it is readily seen that if the T rows of **Y** are orthogonal to one another, $\mathbf{YY}^{\mathrm{T}}$ is a diagonal matrix approximately equal to $pM\mathbf{I}$ (**I** is the identity matrix), so that $z(\mathbf{YY}^{\mathrm{T}}\mathbf{W}) = \mathbf{W}$ and recall is perfect. Notice that the rows of **Y** for the sample memory are nearly orthogonal to one another, and that the purpose of addressing through **A** is to produce (nearly) orthogonal activation vectors for most pairs of addresses, which is a way of saying that the sets of locations activated by dissimilar addresses overlap as little as possible.

3.4 Relation to Correlation-Matrix Memories

The $M \times U$ inner products that make up **C** are correlations of a sort: they are unnormalized correlations that reflect agreement between the M variables represented by the columns of **Y**, and the U variables represented by the columns of **W**. If the columns were normalized to zero mean and to unit length, their inner products would equal the correlation coefficients used commonly in statistics. Furthermore, the inner products of activation vectors (i.e., unnormalized correlations) $\mathbf{Y}_t \cdot \mathbf{y}$ serve as weights for the training words in memory retrieval, further justifying the term correlation-matrix memory.

The **Y**-variables are derived from the **X**-variables (each **Y**-variable compares the data addresses **X** to a specific hard address), whereas in the original correlation-matrix memories (Anderson 1968; Kohonen 1972), the **X**-variables are used directly, and the variables are continuous. Changing from the **X**-variables to the **Y**-

variables means, mathematically, that the input dimension is blown way up (from a thousand to a million); in practice it means that the memory can be made arbitrarily large, rendering its capacity independent of the input dimension N. The idea of expanding the input dimension goes back at least to Rosenblatt's (1962) α-perceptron network.

3.5 Recall Fidelity (φ)

We will now look at the retrieval of words stored in memory, that is, how faithfully are the stored words reconstructed by the retrieval procedure. The asymptotic behavior of the memory, as the input dimension N grows without bound, has been analyzed in depth by Chou (1989). Specific dimension N is assumed here, and the analysis is simple but approximate. The analysis follows one given by Jaeckel (1989a) and uses some of the same symbols.

What happens when we use one of the addresses, say, the last data address $\mathbf{X}_T$, to retrieve a word from memory; how close to the stored word $\mathbf{W}_T$ is the retrieved word $\mathbf{Z}_T$? The output word $\mathbf{Z}_T$ is gotten from the sum vector $\mathbf{S}_T$ by comparing its U sums to zero. Therefore, we need to find out how likely will a sum in $\mathbf{S}_T$ be on the correct side of zero. Since the data are uniform random, all columns of $\mathbf{C}$ have the same statistics, and all sums in $\mathbf{S}_T$ have the same statistics. So it suffices to look at a single coordinate of the data words, say, the last, and to assume that the last bit of the last data word, $\mathbf{W}_{T,U}$, is 1. How likely is $\mathbf{S}_{T,U} > 0$ if $\mathbf{W}_{T,U} = 1$? This likelihood is called the *fidelity* for a single bit, denoted here by φ (phi for 'fidelity'), and we now proceed to estimate it.

The sum vector $\mathbf{S}_T$ retrieved by the address $\mathbf{X}_T$ is a sum over the locations activated by $\mathbf{X}_T$. The locations are indicated by the 1s of the activation vector $\mathbf{Y}_T$, so that $\mathbf{S}_T = \mathbf{Y}_T{}^{\mathrm{T}}\mathbf{C}$, which equals $\mathbf{Y}_T{}^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}\mathbf{W}$ (that $\mathbf{C} = \mathbf{Y}^{\mathrm{T}}\mathbf{W}$ was shown above). The last coordinate of the sum vector is then $\mathbf{S}_{T,U} = \mathbf{Y}_T{}^{\mathrm{T}}\mathbf{C}_{\cdot,U} = \mathbf{Y}_T{}^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}\mathbf{W}_{\cdot,U} = (\mathbf{Y}\mathbf{Y}_T)^{\mathrm{T}}\mathbf{W}_{\cdot,U} = (\mathbf{Y}\mathbf{Y}_T) \cdot \mathbf{W}_{\cdot,U}$, which shows that only the last bits of the data words contribute to it. Thus, the Uth bit-sum is the (inner) product of two vectors, $\mathbf{Y}\mathbf{Y}_T$ and $\mathbf{W}_{\cdot,U}$, where the T-vector $\mathbf{W}_{\cdot,U}$ consists of the stored bits (the last bit of each stored word), and the T components of $\mathbf{Y}\mathbf{Y}_T$ act as weights for the stored bits.

The weights $\mathbf{Y}\mathbf{Y}_T$ have a clear interpretation in terms of activation sets and their intersections or overlaps: they equal the sizes of the overlaps. This is illustrated in Figure 2–6 (cf. Fig. 2–5). For example, since the 1s of $\mathbf{Y}_t$ and $\mathbf{Y}_T$ mark the locations activated by $\mathbf{X}_t$ and $\mathbf{X}_T$, respectively, the weight $\mathbf{Y}_t \cdot \mathbf{Y}_T$ for the tth data word in the sum $\mathbf{S}_T$ equals the number of locations activated by both $\mathbf{X}_t$ and $\mathbf{X}_T$. Because the addresses are uniform random, this overlap is p^2M locations on the average, where p is the probability of activating a location, except that for $t = T$ the two activation sets are the same and the overlap is complete, covering pM locations on the average.

((**FIGURE** 2–6. Activation overlaps as weights for stored words.))

In computing fidelity, we will abbreviate notation as follows: Let B_t (= $\mathbf{W}_{t,U}$) be the last bit of the tth data word, let $L_t = \mathbf{Y}_t \cdot \mathbf{Y}_T$ be its weight in the sum $\mathbf{S}_{T,U}$, and let Σ (= $\mathbf{S}_{T,U}$) be the last bit sum. Regard the bits B_t and their weights L_t as two sets of T random variables, and recall our assumption that addresses and data are uniform random. Then the bits B_t are independent −1s and 1s with equal probability (i.e., mean $\mathrm{E}\{B_t\} = 0$), and they are also independent of the weights. The weights L_t, being sizes of activation overlaps, are nonnegative integers. When activation is low, as it is in the sample memory ($p = 0.000445$), the weights resemble independent Poisson variables: the first $T - 1$ of them have a mean (and variance $\mathrm{Var}\{L_t\} \approx$) $\mathrm{E}\{L_t\} = \lambda_t = \lambda = p^2M$ and the last has a mean (and variance $\mathrm{Var}\{L_T\} \approx$) $\mathrm{E}\{L_T\} = \lambda_T = \Lambda = pM$ (i.e., complete overlap). For the sample memory these values are: mean activation $\Lambda = pM = 445$ locations (out of a million), and mean activation overlap $\lambda = p^2M = 0.2$ locations ($t < T$). We will proceed as if the weights L_t were independent Poisson variables, and hence our result will be approximate.

We are assuming that the bit we are trying to recover equals 1 (i.e., $B_T = \mathbf{W}_{T,U} = 1$); by symmetry, the analysis of $B_T = -1$ is equivalent. The sum Σ is then the sum of T products L_tB_t, and its mean, or expectation, is

$$\mu = \mathrm{E}\{\Sigma\} = \sum_{t=1}^{T-1} \mathrm{E}\{L_tB_t\} + \mathrm{E}\{L_T \cdot 1\}$$

$$= \mathrm{E}\{L_T\}$$

$$= \Lambda$$

because independence and $\mathrm{E}\{B_t\} = 0$ yield $\mathrm{E}\{L_tB_t\} = 0$ when $t < T$. The mean sum can be interpreted as follows: it contains all Λ (= 445) copies of the target bit B_T that have been stored and they reinforce each other, while the other $(T - 1)\lambda$ (= 2,000) bits in Σ tend to cancel out each other.

Retrieval is correct when the sum Σ is greater than 0. However, random variation can make $\Sigma \le 0$. The likelihood of that happening, depends on the variance σ^2 of the sum, which variance we will now estimate. When the terms are approximately independent, their variances are approximately additive, so that

$$\sigma^2 = \mathrm{Var}\{\Sigma\} \approx (T - 1)\mathrm{Var}\{L_1B_1\} + \mathrm{Var}\{L_T \cdot 1\}$$

The second variance is simply $\mathrm{Var}\{L_T\} \approx \Lambda$. The first variance can be rewritten as

$$\mathrm{Var}\{L_1B_1\} \equiv \mathrm{E}\{L_1{}^2B_1{}^2\} - (\mathrm{E}\{L_1B_1\})^2$$

$$= \mathrm{E}\{L_1{}^2\}$$

because $B_1{}^2 = 1$, and because $\mathrm{E}\{L_1B_1\} = 0$ as above. It can be rewritten further as

$$\equiv \mathrm{Var}\{L_1\} + (\mathrm{E}\{L_1\})^2$$

$$\approx \lambda + \lambda^2$$

and we get, for the variance of the sum,

$$\sigma^2 \approx (T-1)\,(\lambda + \lambda^2) + \Lambda$$

Substituting p^2M for λ and pM for Λ, approximating $T-1$ with T, and rearranging finally yields

$$\sigma^2 = \mathrm{Var}\{\Sigma\} \approx pM[1 + pT(1 + p^2M)]$$

We can now estimate the probability of incorrect recall, that is, the probability that $\Sigma \le 0$ when $B_T = 1$. We will use the fact that if the products L_tB_t are T independent random variables, their sum Σ tends to the normal (Gaussian) distribution with mean and variance equal to those of Σ. We then get, for the probability of a single-bit failure,

$$\Pr\{\Sigma \le 0 \mid \mu, \sigma\} \approx \Phi(-\mu/\sigma)$$

where Φ is the normal distribution function; and for the probability of recalling a bit correctly, or bit-fidelity φ, we get $1 - \Phi(-\mu/\sigma)$, which equals $\Phi(\mu/\sigma)$.

3.6 Signal (μ), Noise (σ), and Probability of Activation (p)

We can regard the mean value μ ($=pM$) of the sum Σ as signal, and the variance σ^2 ($\approx pM[1 + pT(1 + p^2M)]$) of the sum as noise. The standard quantity $\rho = \mu/\sigma$ is then a *signal-to-noise ratio* (rho for 'ratio') that can be compared to the normal distribution, to estimate bit-fidelity, as was done above:

$$\varphi = \Pr\{\text{bit recalled correctly}\} \approx \Phi(\rho)$$

The higher the signal-to-noise ration, the more likely are stored words recalled correctly. This points to a way to find good values for the probability p of activating locations and, hence, for the activation radius H: We want p that maximizes ρ. To find this value of p, it is convenient to start with the expression for ρ^2 and to reduce it to

$$\rho^2 = \mu^2/\sigma^2 \approx \frac{pM}{1 + pT\,(1 + p^2M)}$$

Taking the derivative with respect to p, setting it to 0, and solving for p gives

$$p = \frac{1}{\sqrt[3]{2MT}}$$

as the best probability of activation. This value of p was mentioned earlier, and it was used to set parameters for the sample memory.

The probability $p = (2MT)^{-1/3}$ of activating a location is optimal only when exact storage addresses are used for retrieval. When a retrieval address is approximate (i.e., when it equals a storage address plus some noise), both the signal

and the noise are reduced, and also their ratio is reduced. Analysis of this is more complicated than the one above, and it is not carried out here. The result is that, for maximum recovery of stored words with approximate retrieval addresses, p should be somewhat larger than $(2MT)^{-1/3}$ (typically, less than twice as large); however, when the data are clustered rather than uniform random, optimum p tends to be smaller than $(2MT)^{-1/3}$.

In a case yet more general, the training set is not "clean" but contains many noisy copies of each word to be stored, and the data addresses are noisy (cf. Fig. 2–1). Then it makes sense to store words within a smaller radius and to retrieve them within a larger. To allow such memories to be analyzed, Avery Wang (unpublished) and Jaeckel (1988) have derived formulas for the size of the overlap of activation sets with different radii of activation. As a rule, the overlap decreases rapidly with increasing distance between the centers of activation.

3.7 Memory Capacity (τ)

Storage and retrieval in a standard random-access memory are deterministic. Therefore, its capacity (in words) can be expressed simply as the number of memory locations. In a sparse distributed memory, retrieval of words is statistical. However, its capacity, too, can be defined as a limit on the number T of words that can be stored and retrieved successfully, although the limit depends on what we mean by success.

A simple criterion of success is that a stored bit is retrieved correctly with high probability φ (e.g., $0.99 \le \varphi \le 1$). Other criteria can be derived from it or are related to it. Specifically, capacity here is the maximum T, T_{max}, such that $\Pr\{\mathbf{Z}_{t,u} = \mathbf{W}_{t,u}\} \ge \varphi$; we are assuming that exact storage addresses are used to retrieve the words. It is convenient to relate capacity to memory size M and to define it as $\tau = T_{max}/M$. As fidelity φ approaches 1, capacity τ approaches 0, and the values of τ that concern us here are smaller than 1. We will now proceed to estimate τ.

In Section 3.5 on Recall Fidelity we saw that the bit-recall probability φ is approximated by $\Phi(\rho)$, where ρ is the si~~2.1 Random-Access Memory~~gnal-to-noise ratio as defined above. By writing out ρ and substituting τM for T we get

$$\varphi \approx \Phi(\rho) \approx \Phi\left(\left[\frac{pM}{1 + p\tau M(1 + p^2 M)}\right]^{1/2}\right)$$

which leads to

$$[\Phi^{-1}(\varphi)]^2 \approx \rho^2 \approx \frac{pM}{1 + p\tau M(1 + p^2 M)}$$

where Φ^{-1} is the inverse of the normal distribution function. Dividing by pM in the

numerator and the denominator gives

$$[\Phi^{-1}(\varphi)]^2 \approx \frac{1}{\frac{1}{pM} + \tau(1 + p^2 M)}$$

The right side goes to $1/\tau$ as the memory size M grows without bound, giving us a simple expression for the asymptotic capacity:

$$\tau \approx \frac{1}{[\Phi^{-1}(\varphi)]^2}$$

To verify this limit, we use the optimal probability of activation, taking note that it depends on both M and τ: $p = (2MT)^{-1/3} = (2\tau M^2)^{-1/3}$. Then, in the expression above, $1/(pM) = (2\tau/M)^{1/3}$ and goes to zero as M goes to infinity, because $\tau < 1$. Similarly, $\tau(1 + p^2 M) = \tau + (\frac{1}{4}\tau/M)^{1/3}$ and goes to τ.

To compare this asymptotic capacity to the capacity of a finite memory, consider $\varphi = 0.999$, meaning that about one bit in a thousand is retrieved incorrectly. Then the asymptotic capacity is $\tau \approx 0.105$, and the capacity of the million-location sample memory is 0.096. Keeler (1988) has shown that the sparse distributed memory and the binary Hopfield net trained with the outer-product leaning rule, which is equivalent to a correlation-matrix memory, have the same capacity per storage element or counter. The $0.15N$ capacity of the Hopfield net ($\tau = 0.15$) corresponds to fidelity $\varphi = 0.995$, meaning that about one bit in 200 is retrieved incorrectly. The practical significance of the sparse distributed memory design is that, by virtue of the hard locations, the number of storage elements is independent of the input and output dimensions. Doubling the hardware doubles the number of words of a given size that can be stored, whereas the capacity of the Hopfield net is limited by the word size.

A very simple notion of capacity has been used here, and it results in capacities of about 10 percent of memory size. However, the assumption has been that exact storage addresses are used in retrieval. If approximate addresses are used, and if less error is tolerated in the words retrieved than in the addresses used for retrieving them, the capacity goes down. The most complete analysis of capacity under such general conditions has been given by Chou (1989). Expressing capacity in absolute terms, for example, as Shannon's information capacity, is perhaps the most satisfying. This approach has been taken by Keeler (1988). Allocating the capacity is then a separate issue: whether to store many words or to correct many errors. A practical guide is that the number of stored words should be from 1 to 5 percent of memory size (the number of hard locations).

4. SDM as an Artificial Neural Network

The sparse distributed memory, as an artificial neural network, is a synchronous, fully connected, three-layer (or two-layer, see below), feed-forward net illustrated by Figure 2–7. The flow of information in the figure is from left to right. The column of N circles on the left is called the *input* layer, the column of M circles in the middle is called the *hidden* layer, and the column of U circles on the right is called the *output* layer, and the circles in the three columns are called input units, hidden units, and output units, respectively.

((**FIGURE** 2–7. Feed-forward artificial neural network.))

The hidden units and the output units are *bona fide* artificial neurons, so that, in fact, there are only two layers of "neurons." The input units merely represent the outputs of some other neurons. The inputs $\mathbf{x}_n$ to the hidden units label the input layer, the input coefficients $\mathbf{A}_{m,n}$ of the hidden units label the lines leading into the hidden units, and the outputs $\mathbf{y}_m$ of the hidden units label the hidden layer. If y is the activation function of the hidden units (e.g., $y(d) = 1$ if $d \geq D$, and $y(d) = 0$ otherwise), the output of the mth hidden unit is given by

$$\mathbf{y}_m = y\left(\sum_{n=1}^{N} \mathbf{A}_{m,n}\mathbf{x}_n\right)$$

which, in vector notation, is $\mathbf{y}_m = y(\mathbf{A}_m \cdot \mathbf{x})$, where $\mathbf{x}$ is the vector of inputs to, and $\mathbf{A}_m$ is the vector of input coefficients of, the mth hidden unit.

A similar description applies to the output units, with the outputs of the hidden units serving as their inputs, so that the output of the uth output unit is given by

$$\mathbf{z}_u = z\left(\sum_{m=1}^{M} \mathbf{C}_{m,u}\mathbf{y}_m\right)$$

or, in vector notation, $\mathbf{z}_u = z(\mathbf{C}_{\cdot,u} \cdot \mathbf{y})$. Here, $\mathbf{C}_{\cdot,u}$ is the vector of input coefficients of the uth output unit, and z is the activation function.

From the equations above it is clear that the input coefficients of the hidden units form the address matrix $\mathbf{A}$, and those of the output units form the contents matrix $\mathbf{C}$, of a sparse distributed memory. In the terminology of artificial neural nets, these are the matrices of connection strengths (synaptic strengths) for the two layers. 'Fully connected' means that all elements of these matrices can assume nonzero values. Later we will see sparsely connected variations of the model.

Correspondence between Figures 2–7 and 2–4 is now demonstrated by transforming Figure 2–7 according to Figure 2–8, which shows four ways of drawing artificial neurons. View A shows how they appear in Figure 2–7. View B is laid out similarly, but all labels now appear in boxes and circles. In view C, the

diamond and the circle that represent the inner product and the output, respectively, appear below the column of input coefficients, so that these units are easily stacked side by side. View *D* is essentially the same as view *C*, for stacking units on top of each another. We will now redraw Figure 2–7 with units of type *D* in the hidden layer and with units of type *C* in the output layer. An input (a circle) that is shared by many units is drawn only once. The result is Figure 2–9. Its correspondence to Figure 2–4 is immediate, the vectors and the matrices implied by Figure 2–7 are explicit, and the cobwebs of Figure 2–7 are gone.

((**FIGURE** 2–8. Four views of an artificial neuron.))

((**FIGURE** 2–9. Sparse distributed memory as an artificial ...))

In describing the memory, the term 'synchronous' means that all computations are completed in what could be called a machine cycle, after which the network is ready to perform another cycle. The term is superfluous if the net is used as a feed-forward net akin to a random-access memory. However, it is meaningful if the network's output is fed back as input: the network is allowed to settle with each input so that a completely updated output is available as the next input.

As a multilayer feed-forward net, the sparse distributed memory is akin to the nets trained with the error back-propagation algorithm (Rumelhart and McClelland 1986). How are the two different? In a broad sense they are not: we try to find matrices **A** and **C**, and activation functions *y* and *z*, that fit the source of our data. In practice, many things are done differently.

In error back-propagation, the matrices **A** and **C** and the activation vector **y** are usually real-valued, the components of **y** usually range over the interval [−1, 1] or [0, 1], the activation function *y* and its inverse are differentiable, and the data are stored using a uniform algorithm to change both **A** and **C**. In sparse distributed memory, the address matrix **A** is usually binary, and various methods are used to choose it, but once a location's address has been set, it is not changed as the data are stored (**A** is constant); furthermore, the activation function *y* is a step function that yields an activation vector **y** that is mostly 0s, with a few 1s. Another major difference is in the size of the hidden layer. In back-propagation nets, the number of hidden units is usually smaller than the number of input units or the number of items in the training set; in a sparse distributed memory, it is much larger.

The differences imply that, relative to back-propagation nets, the training of a sparse distributed memory is fast (it is easy to demonstrate single-trial learning), but applying it to a new problem is less automatic (it requires choosing an appropriate data representation, as discussed in the section on SDM research below).

5. SDM as a Model of the Cerebellum

5.1 Modeling Biology with Artificial Neural Networks

Biological neurons are cells that process signals in animals and humans, allowing them to respond rapidly to the environment. To achieve speed, neurons use electrochemical mechanisms to generate a signal (a voltage level or electrical pulses) and to transmit it to nearby and distant sites.

Biological neurons come in many varieties. The peripheral neurons couple the organism to the world. They include the sensory neurons that convert an external stimulus into an electrical signal, the motor neurons whose electrical pulses cause muscle fibers to contract, and other effector neurons that regulate the secretion of glands. However, most neurons in highly evolved animals are interneurons that connect directly to other neurons rather than to sensors or to effectors. Interneurons also come in many varieties and they are organized into a multitude of neural circuits.

A typical interneuron has a cell body and two kinds of arborizations: a dendrite tree that receives signals from other neurons, and an axon tree that transmits the neuron's signal to other neurons. Transmission-contact points between neurons are called synapses. They are either excitatory (positive synaptic weight) or inhibitory (negative synaptic weight) according to whether a signal received through the synapse facilitates or hinders the activation of the receiving neuron. The axon of one neuron can make synaptic contact with the dendrites and cell bodies of many other neurons. Thus, a neuron receives multiple inputs, it integrates them, and it transmits the result to other neurons.

Artificial neural networks are networks of simple, interconnected processing units, called *(artificial) neurons*. The most common artificial neuron in the literature has multiple (N) inputs and one output and is defined by a set of input coefficients—a vector of N reals, standing for the synaptic weights—and a nonlinear scalar activation function. The value of this function is the neuron's output, and it serves as input to other neurons. A linear threshold function is an example of an artificial neuron, and the simplest kind—one with binary inputs and output—is used in the sparse distributed memory.

It may seem strange to model brain activity with binary neurons when real neurons are very complex in comparison. However, the brain is organized in large circuits of neurons working in parallel, and the mathematical study of neural nets is aimed more at understanding the behavior of circuits than of individual neurons. An important fact—perhaps the most important—is that the states of a large circuit can be mapped onto the points of a high-dimensional space, so that although a binary neuron is a grossly simplified model of a biological neuron, a large circuit of binary neurons, by virtue of its high dimension, can be a useful model of a circuit of biological neurons.

The sparse distributed memory's connection to biology is made in the standard way. Each row through **A**, **d**, **y**, and **C** in Figure 2–9—each hidden unit—is an artificial neuron that represents a biological neuron. Vector **x** represents the N signals coming to these neurons as inputs from N other neurons (along their axons), vector $\mathbf{A}_m$ represents the weights of the synapses through which the input signals enter the mth neuron (at its dendrites), $\mathbf{d}_m$ represents the integration of the input signals by the mth neuron, and $\mathbf{y}_m$ represents the output signal, which is passed along the neuron's axon to U other neurons through synapses with strengths $\mathbf{C}_m$.

We will call these (the hidden units) the *address-decoder neurons* because they are like the address-decoder circuit of a random-access memory: they select locations for reading and writing. The address that the mth address-decoder neuron decodes is given by the input coefficients $\mathbf{A}_m$; location $\mathbf{A}_m$ is activated by inputs **x** that equal or are sufficiently similar to $\mathbf{A}_m$. How similar, depends on the radius of activation H. It is interesting that a linear threshold function with N inputs, which is perhaps the oldest mathematical model of a neuron, is ideal for address decoding in the sparse distributed memory, and that a proper choice of a single parameter, the threshold, makes it into an address decoder for a location of an ordinary random-access memory.

Likewise, in Figure 2–9, each column through **C**, **s**, and **z** is an artificial neuron that represents a biological neuron. Since these U neurons provide the output of the circuit, they are called the output neurons. The synapses made by the axons of the address-decoder neurons with the dendrites of the output neurons are represented by matrix **C**, and they are modifiable; they are the sites of information storage in the circuit.

We now look at how these synapses are modified; specifically, what neural structures are implied by the memory's storage algorithm (cf. Figs. 2-4 and 2-9). The word **w** is stored by adding it into the counters of the active locations, that is, into the axonal synapses of active address-decoder neurons. This means that if a location is activated for writing, its counters are adjusted upward and downward; if it is not activated, its counters stay unchanged.

Since the output neurons are independent of each other, it suffices to look at just one of them, say, the uth output neuron. See Figure 2-10 center. The uth output neuron produces the uth output bit, which is affected only by the uth bits of the words that have been stored in the memory. Let us assume that we are storing the word **w**. Its uth bit is $\mathbf{w}_u$. To add $\mathbf{w}_u$ into all the active synapses in the uth column of **C**, it must be made physically present at the active synaptic sites of the column. Since different sites in a column are active at different times, it must be made present at all synaptic sites of the column. A neuron's way of presenting a signal is by passing it along the axon. This suggests that the uth bit $\mathbf{w}_u$ of the word-in register should be represented by a neuron that corresponds to the uth output neuron $\mathbf{z}_u$, and

that its output signal should be available at each synapse in column u, although it is "captured" only by synapses that have just been activated by address-decoder neurons **y**. Such an arrangement is shown in Figure 2–10. It suggests that word-in neurons are paired with output neurons, with the axon tree of a word-in neuron possibly meshing with the dendrite tree of the corresponding output neuron, as that would help carry the signal to all synaptic sites of a column. This kind of pairing, when found in a brain circuit, can help us interpret the circuit (Fig. 2-10, on the right).

((**FIGURE** 2–10. Connections to an output neuron.))

5.2 The Cortex of the Cerebellum

Of the neural circuits in the brain, the cortex of the cerebellum resembles the sparse distributed memory the most. The cerebellar cortex of mammals is a fairly large and highly regular structure with an enormous number of neurons of only six major kinds. Its morphology has been studied extensively since early 1900s, its role in fine motor control has been established, and its physiology is still studied intensively (Ito 1984).

The cortex of the cerebellum is sketched in Figure 2–11 after Llinás (1975). Figure 2–12 is Figure 2–9 redrawn in an orientation that corresponds to the sketch of the cerebellar cortex.

((**FIGURE** 2–11. Sketch of the cortex of the cerebellum.))

((**FIGURE** 2–12. Sparse distributed memory's resemblance ...))

Within the cortex are the cell bodies of five kinds of neurons: the granule cells, the Golgi cells, the stellate cells, the basket cells, and the Purkinje cells. Figure 2–11 shows the climbing fibers and the mossy fibers entering and the axons of the Purkinje cells leaving the cortex. This agrees with the two inputs into and the one output from a sparse distributed memory. The correspondence goes deeper: The Purkinje cells that provide the output, are paired with the climbing fibers that provide input. A climbing fiber, which is an axon of an olivary cell that resides in the interior of the cerebellum, could thus have the same role in the cerebellum as the line from a word-in cell through a column of counters has in a sparse distributed memory (see Fig. 2–10), namely, to make a bit of a data word available at a bit-storage site when words are stored.

The other set of inputs enters along the mossy fibers, which are axons of cells outside the cerebellum. They would then be like an address into a sparse distributed memory. The mossy fibers feed into the granule cells, which thus would correspond to the hidden units of Figure 2–12 (they appear as rows across Fig. 2–9) and would perform address decoding. The firing of a granule cell would constitute activating a location for reading or writing. Therefore, the counters of a location would be found

among the synapses of a granule cell's axon; these axons are called parallel fibers. A parallel fiber makes synapses with Golgi cells, stellate cells, basket cells, and Purkinje cells. Since the Purkinje cells provide the output, it is natural to assume that their synapses with the parallel fibers are the storage sites or the memory's counters.

In addition to the "circuit diagram," other things suggest that the cortex of the cerebellum is an associative memory reminiscent of the sparse distributed memory. The numbers are reasonable. The numbers quoted below were compiled by Loebner (1989) in a review of the literature and they refer to the cerebellum of the cat. Several million mossy fibers enter the cerebellum, suggesting that the dimension of the address space is several million. The granule cells are the most numerous—in the billions—implying a memory with billions of hard locations, and only a small fraction of them is active at once, which agrees with the model. Each parallel fiber intersects the flat dendritic trees of several hundred Purkinje cells, implying that a hard location has several hundred counters. The number of parallel fibers that pass through the dendritic tree of a single Purkinje cell is around a hundred-thousand, implying that a single "bit" of output is computed from about a hundred-thousand counters (only few of which are active at once). The number of Purkinje cells is around a million, implying that the dimension of the data words is around a million. However, a single olivary cell sends about ten climbing fibers to that many Purkinje cells, and if, indeed, the climbing fibers train the Purkinje cells, the output dimension is more like a hundred-thousand than a million. All these numbers mean, of course, that the cerebellar cortex is far from fully connected: every granule cell does not reach every Purkinje cell (nor does every mossy fiber reach every granule cell; more on that below).

This interpretation of the cortex of the cerebellum as an associative memory, akin to the sparse distributed memory, is but an outline, and it contains discrepancies that are evident even at the level of cell morphology. According to the model, an address decoder (a hidden unit) should receive all address bits, but a granule cell receives input from three to five mossy fibers only, and for a granule cell to fire, most or all of its inputs must be firing (the number of active inputs required for firing appears to be controlled by the Golgi cells that provide the other major input to the granule cells; the Golgi cells could control the number of locations that are active at once). The very small number of inputs to a granule cell means that activation is not based on Hamming distance from an address but on certain address bits being on in the address register. Activation of locations of a sparse distributed memory under such conditions has been treated specifically by Jaeckel, and the idea is present already in the cerebellar models of Marr and of Albus. These will be discussed in the next two sections.

Many details of the cerebellar circuit are not addressed by this comparison to the

sparse distributed memory. The basket cells connect to the Purkinje cells in a special way, the stellate cells make synapses with the Purkinje cells, and signals from the Purkinje cells and climbing fibers go to the basket cells and Golgi cells. The nature of synapses and signals—the neurophysiology of the cerebellum—has not been considered. Some of these things are addressed by the mathematical models of Marr and of Albus. The point here has been to demonstrate some of the variety in a real neural circuit, to show how a mathematical model can be used to interpret such a circuit, and to suggest that the cortex of the cerebellum constitutes an associative memory. Because its mossy-fiber input comes from all over the cerebral cortex—from many sensory areas—the cerebellum is well located for correlating action that it regulates, with information about the environment.

6. Variations of the Model

The basic sparse distributed memory model is fully connected. This means that every input unit (address bit) is seen by every hidden unit (hard location), and that every hidden unit is seen by every output unit. Furthermore, all addresses and words are binary. If −1 and 1 are used as the binary components, 'fully connected' means that none of the elements of the address and contents matrices **A** and **C** is (identically) zero. Partially—and sparsely—connected models have zeros in one or both of the matrices, as a missing connection is marked by a weight that is zero.

Jaeckel has studied designs with sparse address matrices and binary data. In the selected-coordinate design (1989a), −1s and 1s are assumed to be equally likely in the data addresses; in the hyperplane design (1989b), the data-address bits are assumed to be mostly (e.g., 90%) −1s. This section is based on these two papers. The papers are written in terms of binary 0s and 1s, but here we will use −1s and 1s, and will let 0 stand for a missing connection or a "don't care"-bit (for which Jaeckel uses the value 1/2). Jaeckel uses one-million-location memories (M = 1,000,000) with a 1,000-dimensional address space (N = 1,000) to demonstrate the designs.

6.1 Jaeckel's Selected-Coordinate Design

In the selected- coordinate design, the hard-address matrix **A** has a million rows with ten −1s and 1s (k = 10) in each row. The −1s and 1s are chosen with probability 1/2 and they are placed randomly within the row and independently of other rows; the remaining 990 coordinates of a row are 0s. This is equivalent to taking a uniform random **A** of −1s and 1s and setting a random 990 coordinates in each row to zero (different 990 for different rows). A location is activated if the values of all ten of its selected coordinates match the address register **x**: $\mathbf{y}_m = 1$ iff $\mathbf{A}_m \cdot \mathbf{x} = k$. The probability of activating a hard location is related to the number of nonzero coordinates in a hard address by $p = 0.5^k$. Here, k = 10 and p = 0.001.

6.2 Jaeckel's Hyperplane Design

The hyperplane design deals with data where the addresses are skewed (e.g., 100 1s and 900 −1s). Each row of the hard-address matrix **A** has three 1s ($k = 3$), placed at random, and the remaining 997 places have 0s (there are no −1s). A location is activated if the address register has 1s at those same three places: $\mathbf{y}_m = 1$ iff $\mathbf{A}_m \cdot \mathbf{x} = k$. The probability of activating a location is related to the number of 1s in its address by $p \approx (L/N)^k$, where L is the number of 1s in the data addresses **x**. Here, $N = 1{,}000$, $L = 100$, $k = 3$, and $p \approx 0.001$.

Jaeckel has shown that both of these designs are better than the basic design in recovering previously stored words, as judged by signal-to-noise ratios. They are also easier to realize physically—in hardware—because they require far fewer connections and much less computation in the address-decoder unit that determines the set of active locations.

The region of the address space that activates a hard location in the three designs can be interpreted geometrically as follows: A location of the basic sparse distributed memory is activated by all addresses that are within H Hamming units of the location's address, so that the exciting part of the address space is a hypersphere around the hard address. In the selected coordinate design, a hard location is activated by all addresses in a subspace of the address space defined by the k selected coordinates—that is, by the vertices of an $(N - k)$-dimensional hypercube. In the hyperplane design, the address space is a hyperplane defined by the number of 1s in an address, L (which is constant over all data addresses), and a hard location is activated by the intersection of the address space with the $(N - k)$-dimensional hypercube defined by the k 1s of the hard address.

The regions have a spherical interpretation also in the latter two designs, as suggested by the activation condition $\mathbf{A}_m \cdot \mathbf{x} = k$ (same formula for both designs; see above). It tells that the exciting points of the address space lie on the surface of a hypersphere in Euclidean N-space, with center coordinates $\mathbf{A}_m$ (the hard address) and with Euclidean radius $(N - k)^{1/2}$ (no points of the address space lie inside the sphere). This gives rise to *intermediate designs*, as suggested by Jaeckel (1989b): let the hard addresses be defined in −1s, 0s, and 1s as above, and let the mth hard location be activated by all addresses **x** within a suitably large hypersphere centered at the hard address. Specifically, $\mathbf{y}_m = 1$ if, and only if, $\mathbf{A}_m \cdot \mathbf{x} \geq G$. The parameters G and k (and L) have to be chosen so that the probability of activating a location is reasonable.

The optimum probability of activation p for the various sparse distributed memory designs is about the same—it is in the vicinity of $(2MT)^{-1/3}$—and the reason is that, in all these designs, the sets of locations activated by two addresses, **x** and **x′**, overlap minimally unless **x** and **x′** are very similar to each other. The sets behave in the manner of random sets of approximately pM hard locations each, with

two such sets overlapping by p^2M locations, on the average (unless $\mathbf{x}$ and $\mathbf{x}'$ are very similar to each other). This is a consequence of the high dimension of the address space.

In the preceding section on the cerebellum we saw that the hard-address matrix $\mathbf{A}$, as implied by the few inputs (3–5 mossy fibers) to each granule cell, is very sparse, and that the number of active inputs required for a granule cell to fire, can be modulated by the Golgi cells. This means that the activation of granule cells in the cerebellum resembles the activation of locations in an intermediate design that is close to the hyperplane design.

Not only are the mossy-fiber connections to a granule cell few (3–5 out of several million), but also the granule-cell connections to a Purkinje cell are few (hundred thousand out of billions), so that also the contents matrix $\mathbf{C}$ is very sparse. This aspect of the cerebellum has not been modeled mathematically.

6.3 Hassoun's Pseudorandom Associative Neural Memory

Independently of the above developments, Hassoun (1988) has proposed a model with a random, fixed address matrix $\mathbf{A}$ and variable contents matrix $\mathbf{C}$. This model allows us to extend the concepts of this chapter to data with short addresses (e.g., $N = 4$ bits), and it introduces ideas about storing the data (i.e., training) that can be applied to associative memories at large.

The data addresses $\mathbf{X}$ and words $\mathbf{W}$ in Hassoun's examples are binary vectors in 0s and 1s. The elements of the hard-address matrix $\mathbf{A}$ are small integers; they are chosen at uniform random from the symmetric interval $\{-L, -L+1, -L+2, \ldots, L\}$, where L is a small positive integer (e.g., $L = 3$). Each hard location has its own activation threshold D_m, which is chosen so that approximately half of all possible N-bit addresses $\mathbf{x}$ activate the location: $\mathbf{y}_m = 1$ if $\mathbf{A}_m \cdot \mathbf{x} \geq D_m$, and $\mathbf{y}_m = 0$ otherwise. The effect of such addressing through $\mathbf{A}$ is to convert the matrix $\mathbf{X}$ of N-bit data addresses into the matrix $\mathbf{Y}$ of M-bit activation vectors, where $M \gg N$ and where each activation vector $\mathbf{y}_m$ is about half 0s and half 1s (probability of activation p is around 0.5).

Geometric interpretation of addressing through $\mathbf{A}$ is as follows. The space of hard addresses is an N-dimensional hypercube with sides of length $2L + 1$. The unit cubes or cells of this space are potential hard locations. The M hard addresses $\mathbf{A}_m$ are chosen at uniform random from within this space. The space of data addresses is an N-cube with sides of length 2; it is at the center of the hard-address space, with the cell 000...0 at the very center. The data addresses that activate the location $\mathbf{A}_m$ are the ones closest to $\mathbf{A}_m$ and they can be visualized as follows: A straight line is drawn from $\mathbf{A}_m$ through 000...0. Each setting of the threshold D_m then corresponds to an $N - 1$-dimensional hyperplane perpendicular to this line, at some distance from $\mathbf{A}_m$. The cells $\mathbf{x}$ of the data-address space that are on the $\mathbf{A}_m$ side of the plane

will activate location $\mathbf{A}_m$. The threshold D_m is chosen so that the plane cuts the data-addresses space into two nearly equal parts.

The hard addresses $\mathbf{A}_m$ correspond naturally to points (and subspaces) $\mathbf{A}'_m$ of the data-address space $\{0, 1\}^N$ gotten by replacing the negative components of $\mathbf{A}_m$ by 0s, the positive components by 1s, and the 0s by either (a "don't care"). The absolute values of the components of $\mathbf{A}_m$ then serve as weights, and the mth location is activated by $\mathbf{x}$ if the *weighted* distance between $\mathbf{A}'_m$ and $\mathbf{x}$ is below a threshold (cf. Kanerva 1988, pp. 46–48).

High probability of activation ($p \approx 0.5$) works poorly with the outer-product leaning rule. However, it is appropriate for an analytic solution to storage by the Ho–Kashyap recording algorithm (Hassoun and Youssef 1989). This algorithm finds a contents matrix $\mathbf{C}$ that solves the linear inequalities implied by $\mathbf{Z} = \mathbf{W}$, where $\mathbf{W}$ is the matrix of data words to be stored, and $\mathbf{Z} = z(\mathbf{S}) = z(\mathbf{YC})$ is the matrix of words retrieved by the rows of $\mathbf{X}$. The inequalities follow from the definition of the threshold function z, as $\mathbf{W}_{t,u} = 1$ implies that $\mathbf{S}_{t,u} > 0$, and $\mathbf{W}_{t,u} = 0$ implies that $\mathbf{S}_{t,u} < 0$. Hassoun and Youssef have shown that this storage algorithm results in large basins of attraction around the data addresses, and that if data are stored autoassociatively, false attractors (i.e., spurious stable patterns and limit cycles) will be relatively few.

6.4 Adaptation to Continuous Variables by Prager and Fallside

All the models discussed so far have had binary vectors as inputs and outputs. Prager and Fallside (1989) consider several ways of extending the sparse distributed memory model into real-valued inputs. The following experiment with spoken English illustrates their approach.

Eleven vowels were spoken several times by different people. Each spoken instance of a vowel is represented by a 128-dimensional vector of reals that serves as an address or cue. The corresponding data word is an 11-bit label. One of the bits in a label is a 1, and its position corresponds to the vowel in question. This is a standard setup for classification by artificial neural nets.

For processing on a computer, the input variables are discretized into 513 integers in the range 16,127–16,639. The memory is constructed by choosing (2,000) hard addresses at uniform random from a 128-dimensional hypercube with sides of length 32,768. The cells of this outer space are addressed naturally by 128-place integers to base 32,768 (i.e., these are the vectors $\mathbf{A}_m$), and the data addresses $\mathbf{x}$ then occupy a small hypercube at the center of the hard-address space; the data-address space is a 128-dimensional cube with sides of length 513. Activation is based on distance. Address $\mathbf{x}$ activates the mth hard location if the "city-block" (L_1) distance between $\mathbf{x}$ and $\mathbf{A}_m$ is at most 16,091. About ten percent of the hard locations will be activated. Experiments with connected speech deal similarly with 896-dimensional real vectors.

Prager and Fallside train the contents matrix **C** iteratively by correcting errors so as to solve the inequalities implied by **Z** = **W** (see the last paragraph of Sec. 6.3).

This design is similar to Hassoun's design discussed in Section 6.3, in that both have a large space of hard addresses that includes, at the center, a small space of data addresses, and that the hard locations are placed at random within the hard-address space. The designs are in contrast with Albus' CMAC (discussed in the next section), where the placement of the hard locations is systematic. Furthermore, the number of input variables in CMAC is small compared to the numbers used by Prager and Fallside.

7. Relation to the Cerebellar Models of Marr and of Albus

The first comprehensive mathematical models of the cerebellum as an associative memory are by Marr (1969) and by Albus (1971), developed independently in their doctoral dissertations, and they still are the most complete of any such models. They were developed specifically as models of the cerebellar cortex, whereas the sparse distributed memory's resemblance to the cerebellum was noticed only after the model had been developed fully.

Marr's and Albus's models attend to many of the details of the cerebellar circuit. The models are based mostly on connectivity but also on the nature of the synapses. Albus (1989) has made a comparison of the two models. The models will be described here insofar as to show their relation to the sparse distributed memory.

7.1 Marr's Model of the Cerebellum

The main circuit in Marr's model—in Marr's terminology and in our symbols—consists of (N =) 7,000 input fibers that feed into (M =) 200,000 codon cells that feed into a single output cell. The input fibers activate codon cells, and codon-cell connections with the output cell store information. The correspondence to the cerebellum is straightforward: the input fibers model mossy fibers, the codon cells model granule cells, and the output cell models a Purkinje cell.

Marr discusses at length the activation of codon cells by the input fibers. Since the input fibers represent mossy fibers and the codon cells represent granule cells, each codon cell receives input from 3–5 fibers in Marr's model. The model assumes discrete time intervals. During an interval an input fiber is either inactive (−1) or active (+1), and at the end of the interval a codon cell is either inactive (0) or active (+1) according to the activity of its inputs during the interval; the codon-cell output is a linear threshold function of its inputs, with +1 weights.

The overall pattern of activity of the N input fibers during an interval is called the input pattern (an N-vector of −1s and 1s), and the resulting pattern of activity of the M codon cells at the end of the interval is called a codon representation of the input pattern (an M-vector of 0s and 1s). These correspond, respectively, to the

address register **x**, and to the activation vector **y**, of a sparse distributed memory.

Essential to the model is that M is much larger than N, and that the number of 1s in a codon representation is small compared to M, and relatively constant; conditions that hold also for the sparse distributed memory. Then the codon representation amplifies differences between input patterns. To make differences in N-bit patterns commensurate with differences in M-bit patterns, Marr uses a relative measure defined as the number of 1s that two patterns have in common, divided by the number of places where either pattern has a 1 (i.e., the size of the intersection of 1s relative to the size of their union).

Relation to artificial neural networks is simple. The input fibers correspond to input units, the codon cells correspond to hidden units, and the output cell corresponds to an output unit. Each hidden unit has only 3–5 inputs, chosen at random from the N input units, and the input coefficients are fixed at +1. Obviously, the net is far from fully connected, but all hidden units are connected to the output unit, and these connections are modifiable. The hidden units are activated by a linear threshold function, and the threshold varies. However, it varies not as the result of training but dynamically so as to keep the number of active hidden units within desired limits (500–5,000). Therefore, to what first looks like a feed-forward net must be added feedback connections that adjust dynamically the thresholds of the hidden units. The Golgi cells are assumed to provide this feedback.

In relating Marr's model to the sparse distributed memory, the codon cells correspond to hard locations, and the hard-address matrix **A** is very sparse, as each row has k_m 1s ($k_m = 3, 4, 5$), placed at random, and $N - k_m$ 0s (there are no −1s in **A**). A codon cell fires if most of its 3–5 inputs are active, and the Golgi cells set the firing threshold so that 500–5,000 codon cells (out of the 200,000) are active at any one time, regardless of the number of active input lines. Thus, the activation function y_m for hard location $\mathbf{A}_m$ is a threshold function with value 1 (the codon cell fires) when most—but not necessarily all—of the k_m 1s of $\mathbf{A}_m$ are matched by 1s in the address **x**. The exact condition of activation in the examples developed by Marr is that $\mathbf{A}_m \cdot \mathbf{x} \geq R$, where the threshold R is between 1 and 5 and depends on **x**. Thus, the codon cells are activated in Marr's model in a way that resembles the activation of hard locations in an intermediate design of sparse distributed memory that is close to the hyperplane design (in the hyperplane design, *all* inputs must be active for a cell to fire).

One of the conditions of the hyperplane design is far from being satisfied—namely, that the number of 1s in the address is about constant (hence the name hyperplane design). In Marr's model it is allowed to vary widely (between 20 and 1,000 out of 7,000), and this creates the need for adjusting the threshold dynamically. In the sparse distributed memory variations discussed so far, the threshold is fixed, but later in this chapter we will refer to experiments in which the

thresholds are adjusted either dynamically or by training with data.

Marr estimates the capacity of his model under the most conservative of assumptions, namely, that (0s and) 1s are added to one-bit counters that are initially 0. Under this assumption, all counters eventually saturate and all information is lost, as pointed out by Albus (1989).

7.2 Albus' Cerebellar Model Arithmetic Computer (CMAC)

This description of CMAC is based on the one in Albus' book *Brains, Behavior, and Robotics* (1981) and uses its symbols. The purpose here is to describe it sufficiently to allow its comparison to the sparse distributed memory.

CMAC is an associative memory with a large number of addressable storage locations, just as the sparse distributed memory is, and the address space is multidimensional. However, the number of dimensions, N, is relatively small (e.g., $N = 14$), while each dimension, rather than being binary, spans a discrete range of values $\{0, 1, 2, \ldots, R - 1\}$. The dimensions are also called input variables, and an input variable might represent a joint angle of a robot arm (0–180 degrees) discretized in five-degree increments (resolution $R = 36$), and a 14-dimensional address might represent the angular positions and velocities of the joints in a seven-jointed robot arm. Different dimensions can have different resolutions, but we assume here, for simplicity, that all have the same resolution R.

An N-dimensional address in this space can be represented by an N-dimensional unit cube, or *cell*, and the entire address space is then represented by R^N of these cells packed into an N-dimensional cube with sides of length R. The cells are addressed naturally by N-place integers to base R.

A storage location is activated by some addresses and not by others. In the sparse distributed memory, these exciting addresses occupy an N-dimensional sphere with Hamming radius H, centered at the location's address. The exciting region of the address space in Albus' CMAC is an N-dimensional cube with sides of length K ($1 < K < R$); it is a *cubicle* of K^N cells (near the edge of the space it is the intersection of such a cubicle with the address space and thus contains fewer than K^N cells). The center coordinates of the cubicle can be thought of as the location's address (the center coordinates are integers if K is odd and half-way between two integers if K is even, and the center can lie outside the R^N cube).

The hard locations of a sparse distributed memory are placed randomly in the address space; those of CMAC—the cubicles—are arranged systematically as follows: First, the R^N cube is packed with the K^N cubicles starting from the corner cell at the origin—the cell addressed by $(0, 0, 0, \ldots, 0)$. This defines a set of $\lceil R/K \rceil^N$ hard locations (the ceiling of the fraction means that the space is covered completely). The next set of $(1 + \lceil (R - 1)/K \rceil)^N$ hard locations is defined by moving the entire package of cubicles up by one cell along the principle diagonal

of the R^N cube—a translation. To cover the entire address space, cubicles are added next to the existing ones at this stage. This is repeated until K sets of hard locations have been defined (K translations take the cubicles to the starting position), resulting in a total of at least $K\lceil R/K\rceil^N$ hard locations. Since each set of hard locations covers the entire R^N address space, and since the locations in a set do not overlap, each address activates exactly one location in each set and so it activates K locations overall. Conversely, each location is activated by the K^N addresses in its defining cubicle (by fewer if the cubicle spills over the edge of the space). The systematic placement of the hard locations allows addresses to be converted into activation vectors very efficiently in a hardware realization or in a computer simulation (Albus 1980).

Correspondence of the hard locations to the granule cells of the cerebellum is natural in Albus' model. To make the model life-like, each input variable (i.e., each coordinate of the address) is encoded in $R + K - 1$ bits. A bit in the encoding represents a mossy fiber, so that a vector of N input variable (an address) is presented to CMAC as binary inputs on $N(R + K - 1)$ mossy fibers. In the model, each granule cell receives input from N mossy fibers, and each mossy fiber provides input to at least $\lceil R/K\rceil^N$ granule cells.

The 20-bit code for an input variable s_n with range $R = 17$ and with $K = 4$ is given in Table 2–1. It corresponds to the encoding of the variables s_1 and s_2 in Figure 6.8 in Albus' book (1981, p. 149). The bits are labeled with letters above the code in Table 2–1, and the same letters appear below the code in four rows. Bit A, for example, is on (+) when the input variable is at most 3, bit B is on when the input variable falls between 4 and 7, and so forth.

((**TABLE** 2–1. Encoding a 17-level Input Variable s_n ...))

This encoding mimics nature. Many receptor neurons respond maximally to a specific value of an input variable and to values near it. An address bit (a mossy fiber) represents such a receptor, and it is (+)1 when the input variable is near this specific value. For example, this "central" value for bit B is 5.5.

The four rows of labels below the code in Table 2–1 correspond to the four sets of cubicles ($K = 4$) that define the hard locations (the granule cells) of CMAC. The first set depends only on the input bits labeled by the first row. If the code for an input variables s_n has Q_1 first-row bits ($Q_1 = 5$ in Table 2–1), then the NQ_1 first-row bits of the N input variables define ${Q_1}^N$ hard locations by assigning a location to each set of N inputs that combines one first-row bit from each input variable. The second set of ${Q_2}^N$ hard locations is defined similarly by the NQ_2 second-row bits, and so forth with the rest.

We are now ready to describe Albus' CMAC design as a special case of Jaeckel's hyperplane design. The N input variables s_n are encoded and concatenated into an $N(R + K - 1)$-bit address $\mathbf{x}$, which will have NK 1s and $N(R - 1)$ −1s. The

address matrix **A** will have $\sum_k Q_k^N$ rows, and each row will have N 1s, arranged according to the description in the preceding paragraph. The rest of **A** will be 0s (for "don't care"; there will be no −1s in **A**). The activation vector **y** can then be computed as in the hyperplane design: the mth location is activated by **x** if the 1s of the hard address $\mathbf{A}_m$ are matched by 1s in **x** (i.e., iff $\mathbf{A}_m \cdot \mathbf{x} = N$).

After a set of locations has been activated, CMAC is ready to transfer data. Here, as with the sparse distributed memory, we can look at a single coordinate of a data words only, say, the uth coordinate. Since CMAC data are continuous or graded rather than binary, the storage and retrieval rules cannot be identical to those of a sparse distributed memory, but they are similar. Retrieval is simpler: we use the sum $\mathbf{s}_u$ as output and we omit the final thresholding. From the regularity of CMAC it follows that the sum is over K active locations.

From this is derived a storage (learning) rule for CMAC: Before storing the desired output value $\hat{p}_u$ at **x**, retrieve $\mathbf{s}_u$ using **x** as the address and compute the error $\mathbf{s}_u - \hat{p}_u$. If the error is acceptable, do nothing. If the error is too large, correct the K active counters (elements of the matrix **C**) by adding $g(\hat{p}_u - \mathbf{s}_u)/K$ to each, where g $(0 < g \leq 1)$ is a gain factor that affects the rate of learning. This storage rule implies that the counters in **C** count at intervals no greater than one Kth of the maximum allowable error (the counting interval in the basic sparse distributed memory is 1).

In summary, multidimensional input to CMAC can be encoded into a long binary vector that serves as an address to a hyperplane-design sparse distributed memory. The address bits and the hard-address decoders correspond very naturally to the mossy fibers and the granule cells of the cerebellum, respectively, and the activation of a hard location corresponds to the firing of a granule cell. The synapses of the parallel fibers with the Purkinje cells are the storage sites suggested by the model, and the value of an output variable is represented by the firing frequency of a Purkinje cell. Training of CMAC is by error-correction, which presumably is the function of the climbing fibers in the cerebellum.

8. SDM Research

So far in this chapter we have assumed that the hard addresses and the data are a uniform random sample of their respective spaces (the distribution of the hard locations in CMAC is uniform systematic). This has allowed us to establish a base line: we have estimated signal, noise, fidelity, and memory capacity, and we have suggested reasonable values for various memory parameters. However, data from real processes tend to occur in clusters, and large regions of the address space are empty. When such data are stored in a uniformly distributed memory, large numbers of locations are never activated and hence are wasted, and many of the active locations are activated repeatedly so that they, too, are mostly wasted as their contents turn into noise.

There are many ways to counter this tendency of data to cluster. Let us look at the clustering of data addresses first. Several studies have used the memory efficiently by distributing the hard addresses **A** according to the distribution of the data addresses **X**. Keeler (1988) observed that when the two distributions are the same and the activation radius H is adjusted for each storage and retrieval operation so that nearly optimal number of locations are activated, the statistical properties of the memory are close to those of the basic memory with uniformly random hard addresses. In agreement with that, Joglekar (1989) experimented with NETtalk data and got his best results by using a subset of the data addresses as hard addresses (NETtalk transcribes English text into phonemes; Sejnowski and Rosenberg 1986). In a series of experiments by Danforth (1990), recognition of spoken digits, encoded in 240 bits, improved dramatically when uniformly random hard addresses were replaced by addresses that represented spoken words, but the selected-coordinate design with three coordinates performed the best. In yet another experiment, Saarinen et al. (1991b) improved memory utilization by distributing the hard addresses with Kohonen's self-organizing algorithm.

Two studies have shown that uniform random hard addresses can be used with clustered data if the rule for activating locations is adjusted appropriately. In Kanerva (1991), storage and retrieval require two steps: the first to determine a vector of N positive weights for each data address $\mathbf{X}_t$, and the second to activate locations according to a weighted Hamming distance between $\mathbf{X}_t$ and the hard addresses **A**. In Pohja and Kaski (1992), each hard location has its own radius of activation H_m, which is chosen based on the data addresses **X** so that the probability of activating a location is nearly optimal.

It is equally important to deal with clustering in the stored words. For example, some of their bits may be mostly on, some may be mostly off, and some may depend on others. It is possible to analyze the data (**X**, **Z**) and the hard addresses **A** and to determine optimal storage and retrieval algorithms (Danforth 1991), but we can also use iterative training by error correction, as described above for Albus' CMAC. This was done by Joglekar and by Danforth in their above-mentioned experiments. When error correction is used, it compensates for the clustering of addresses as well, but it also introduces the possibility of overfitting the model to the training set.

Two studies by Rogers (1989a, 1990a) deal specifically with the interactions of the data with the hard addresses **A**. In the first of these he concludes that, in computing the sum vector **s**, the active locations should be weighted according to the words stored in them—in fact, each active counter $\mathbf{C}_{m,u}$ might be weighted individually. This would take into account at once the number of words stored in a hard location and the uniformity of those words, so as to give relatively little weight to locations or counters that record mostly noise. In the second study he uses a genetic algorithm to arrive at a set of hard addresses that would store the most

information about a variable in weather data.

Other research issues include the storage of sequences (Manevitz, 1991) and the hierarchical storage of data (Manevitz and Zemach, work in progress).

Most studies of sparse distributed memory have used binary data and have dealt with multivalued variables by encoding them according to an appropriate binary code. Table 2–1 is an example of such a code. Important about the code is that the Hamming distance between codewords corresponds to the difference between the values being encoded (it grows with the difference until a maximum of $2k$ is reached, after which the Hamming distance stays at the maximum).

Sparse distributed memory has been simulated on many computers (Rogers 1990b), including the highly parallel Connection Machine (Rogers 1989b) and special-purpose neural-network computers (Nordström 1991). Hardware implementations have used standard logic circuits and memory chips (Flynn et al. 1987) and programmable gate arrays (Saarinen et al. 1991a).

9. Associative Memory as a Component of a System

In practical systems, an associative memory plays but a part. It can store and recall large numbers of large patterns (high-dimensional vectors) based on other large patterns that serve as memory cues, and it can store and recall long sequences of such patterns, doing it all in the presence of noise. In addition to generating output patterns, the memory provides an estimate of their reliability based on the data it has stored. But that is all; the memory assigns no meaning to the data beyond the reliability estimate. The meaning is determined by other parts of the system, which are also responsible for processing data into forms that are appropriate for an associative memory. Sometimes these other tasks are called preprocessing and postprocessing, but the terms are misleading inasmuch as they imply that preprocessing and postprocessing are minor peripheral functions. They are major functions—at least in the nervous systems of animals they are—and feedback from memory is integral to these "peripheral" functions.

For an example of what a sensory processor must do in producing patterns for an associative memory, consider identifying objects by sight, and assume that the memory is trained to respond with the name of an object, in some suitable code, when presented with an object (i.e., when addressed by the encoding for the object). In what features should objects be encoded? To make efficient use of the memory, all views of an object—past, present, and future—should get the same encoding, and any two different objects should get different encodings. The name, as an encoding, satisfies this condition and so it is an ideal encoding, except that it is arbitrary. What we ask of the visual system is to produce an encoding that reflects physical reality and that can serve as an input to an associative memory, which then outputs the name.

For this final naming step to be successful—even with views as yet unseen—different views of an object should produce encodings that are similar to each other as measured by something like the Hamming distance, but that are dissimilar to the encodings of other objects. A raw retinal image (a pixel map) is a poor encoding, because the retinal cells excited by an object vary drastically with viewing distance and with gaze relative to the object. It is simple for us to fix the gaze—to look directly at the object—but it is impractical to bring objects to a standard viewing distance in order to recognize them. Therefore, the visual system needs to compensate for changes in viewing distance by encoding—by expressing images in features that are relatively insensitive to viewing distance. Orientation of lines in the retinal image satisfy this condition, making them good features for vision. This may explain the abundance of orientation-sensitive neurons in the visual cortex, and why the human visual system is much more sensitive to rotation than to scale (we are poor at recognizing objects in new orientations; we must resort to mental rotation). Encoding shapes in long vectors of bits for an associative memory, where a bit encodes orientation at a location, has been described by Kanerva (1990).

What about the claim that "peripheral" processing, particularly sensory processing, is a major activity in the brain? Large areas of the brain are specific to one sensory modality or another.

In robots that learn, an associative memory stores a world model that relates sensory input to action. The flow of events in the world is presented to the memory as a sequence of large patterns. These patterns encode sensor data, internal-state variables, and commands to the actuators. The memory's ability to store these sequences and to recall them under conditions that resemble the past, allows its use for predicting and planning. Albus (1981, 1991) argues that intelligent behavior of animals and robots in complex environments requires not just one associative memory but a large hierarchy of them, with the sensors and the actuators at the bottom of the hierarchy.

10. Summary

In this chapter we have explored a number of related designs for an associative memory. Common to them is a feed-forward architecture through two layers of input coefficients or weights represented by the matrices **A** and **C**. The matrix **A** is constant, and the matrix **C** is variable. The M rows of **A** are interpreted as the addresses of M hard locations, and the M rows of **C** are interpreted as the contents of those locations. The rows of **A** are a random sample of the hard-address space in all but the Albus' CMAC model, in which the sample is systematic. When the sample is random, it should allow for the distribution of the data.

The matrix **A** and the threshold function y transform N-dimensional input vectors into M-dimensional activation vectors of 0s and 1s. Since M is much larger

than N, the effect is a tremendous increase over the input dimension and a corresponding increase in the separation of patterns and in memory capacity. This simplifies the storage of words by matrix **C**. The training of **C** can be by the outer-product learning rule, by error correction (delta rule), by an analytic solution of a set of linear inequalities, or by a combination of the above. Training, by and large, is fast. These memories require much hardware per stored pattern, but the resolution of the components can be low.

The high fan-out and subsequent fan-in (divergence and convergence) implied by these designs are found also in many neural circuits in the brain. The correspondence is most striking in the cortex of the cerebellum, suggesting that the cerebellum could function as an associative memory with billions of hard locations, each one capable of storing several-hundred-bit words.

The properties of these associative memories imply that if such memory devices, indeed, play an important part in the brain, the brain must also include devices that are dedicated to the sensory systems and that transform sensory signals into forms appropriate for an associative memory.

Pattern Computing. The nervous system offers us a new model of computing, to be contrasted with traditional numeric computing and symbolic computing. It deals with large patterns as computational units and therefore it might be called *pattern computing*. The main units in numeric computing are numbers, say, 32-bit integers or 64-bit floating-point numbers, and we think of them as data; in symbolic computing they are pointers of fewer than 32 bits, and we can think of them as names (very compact, "ideal" encodings; see discussion on sensory encoding in Sec. 9). In contrast, the units in pattern computing have hundreds or thousands of bits, they serve both as pointers and as data, and they need not be precise. Nature has found a way to compute with such units, and we are barely beginning to understand how it is done. It appears that much of the power of pattern computing derives from the geometry of very-high-dimensional spaces and from the parallelism in computing that it allows.

Acknowledgments

This work was supported by the National Aeronautics and Space Administration (NASA) Cooperative Agreement NC2-387 with the Universities Space Research Association (USRA). Computers for the work were a gift from Apple Computer Company. Many of the ideas came from the SDM Research Group of RIACS at the NASA–Ames Research Center. We are indebted to Dr. Michael Raugh for organizing and directing the group.

References

Albus, J.S. 1971. A theory of cerebellar functions. *Mathematical Biosciences* 10:25–61.

Albus, J.S. 1980. Method and Apparatus for Implementation of the CMAC Mapping Algorithm. U.S. Patent No. 4,193,115.

Albus, J.S. 1981. *Brains, Behavior, and Robotics*. Peterborough, N.H.: BYTE/McGraw–Hill.

Albus, J.S. 1989. The Marr and Albus theories of the cerebellum: Two early models of associative memory. *Proceedings COMPCON Spring '89* (34th IEEE Computer Society International Conference), pp. 577–582. Washington, D.C.: IEEE Computer Society Press.

Albus, J.S. 1991. Outline for a theory of intelligence. *IEEE Transactions on Systems, Men, and Cybernetics* 31(3):473–509.

Anderson, J.A. 1968. A memory storage module utilizing spatial correlation functions. *Kybernetik* 5(3):113–119.

Chou, P.A. 1989. The capacity of the Kanerva associative memory. *IEEE Transactions on Information Theory* 35(2):281–298.

Danforth, D. 1990. An empirical investigation of sparse distributed memory using discrete speech recognition. *Proceedings of International Neural Network Conference (Paris)* 1:183–186 (Norwell, Mass.: Kluver Academic). Complete report, with the same title, in RIACS TR 90.18, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Danforth, D. 1991. Total Recall in Distributed Associative Memories. Report RIACS TR 91.3, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Flynn, M.J., P. Kanerva, B. Ahanin, N. Bhadkamkar, P. Flaherty, and P, Hinkley. 1987. Sparse Distributed Memory Prototype: Principles of Operation. Report CSL–TR78–338, Computer Systems Laboratory, Stanford University.

Hassoun, M.H. 1988. Two-level neural network for deterministic logic processing. In N. Peyghambarian (ed.) *Optical Computing and Nonlinear Materials* (Proceedings SPIE 881:258–264).

Hassoun, M.H., and A.M. Youssef. 1989. High performance recording algorithm for Hopfield model associative memories. *Optical Engineering* 28(1):46–54.

Hopfield, J.J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences (Biophysics)* 79(8):2554–2558. Reprinted in J.A. Anderson and E. Rosenfeld (eds.), *Neurocomputing: Foundations of Research*, pp. 460–464 (Cambridge, Mass.: MIT Press).

Ito, M. 1984. *The Cerebellum and Neuronal Control*. New York: Raven Press.

Jaeckel, L.A. 1988. Two Alternate Proofs of Wang's Lune Formula for Sparse Distributed Memory and an Integral Approximation. Report RIACS TR 88.5, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Jaeckel, L.A. 1989a. An Alternative Design for a Sparse Distributed Memory. Report RIACS TR 89.28, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Jaeckel, L.A. 1989b. A Class of Designs for a Sparse Distributed Memory. Report RIACS TR 89.30, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Joglekar, U.D. 1989. Learning to Read Aloud: A Neural Network Approach Using Sparse Distributed Memory. Master's thesis, Computer Science, UC Santa Barbara. Reprinted as report RIACS TR 89.27, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Kanerva, P. 1988. *Sparse Distributed Memory*. Cambridge, Mass.: Bradford/MIT Press.

Kanerva, P. 1990. Contour-map encoding of shape for early vision. In D.S. Touretzky (ed.), *Neural Information Processing Systems* 2:282–289 (Proceedings NIPS–89) (San Mateo, Calif.: Kaufmann).

Kanerva, P. 1991. Effective packing of patterns in sparse distributed memory by selective weighting of input bits. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (eds.), *Artificial Neural Networks* 1:279–284 (Proceedings ICANN–91) (Amsterdam: Elsevier/North–Holland).

Keeler, J.D. 1988. Comparison between Kanerva's SDM and Hopfield-type neural networks. *Cognitive Science* 12:299–329.

Kohonen, T. 1972. Correlation matrix memories. *IEEE Transactions on Computers* C 21(4):353–359. Reprinted in J.A. Anderson and E. Rosenfeld (eds.), *Neurocomputing: Foundations of Research*, pp. 174–180 (Cambridge, Mass.: MIT Press).

Kohonen, T. 1980. *Content-Addressable Memories*. New York: Springer–Verlag.

Kohonen, T. 1984. *Self-Organization and Associative Memory*, second edition. New York: Springer–Verlag.

Kohonen, T., and Reuhkala, E. 1978. A very fast associative method for the recognition and correction of misspelt words, based on redundant hash addressing. *Proceedings of the Fourth International Joint Conference on Pattern Recognition*, Kyoto, Japan, pp. 807–809.

Llinás, R.R. 1975. The cortex of the cerebellum. *Scientific American* 232(1):56–71.

Loebner, E.E. 1989. Intelligent network management and functional cerebellum synthesis. *Proceedings COMPCON Spring '89* (34th IEEE Computer Society International Conference), pp. 583–588. Washington, D.C.: IEEE Computer Society Press. Reprinted in *The Selected Papers of Egon Loebner* (Palo Alto: Hewlett Packard Laboratories, 1991, pp. 205–209).

Manevitz, L.M. 1991. Implementing a "sense of time" via entropy in associative memories. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (eds.), *Artificial Neural Networks* 2:1211–1214 (Proceedings ICANN–91) (Amsterdam: Elsevier/North–Holland).

Manevitz, L.M., and Zemach, Y. Assigning Meaning to Data: Multilevel Information Processing in Kanerva's SDM. Work in progress.

Marr, D. 1969. A theory of cerebellar cortex. *Journal of Physiology (London)* 202:437–470.

Nordström, T. 1991. Designing and Using Massively Parallel Computers for Artificial Neural Networks. Licentiate thesis 1991:12L, Luleå University of Technology, Sweden.

Pohja, S., and K. Kaski. 1992. Kanerva's Sparse Distributed Memory with Multiple Hamming Thresholds. Report RIACS TR 92.06, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Prager, R.W., and F. Fallside. 1989. The modified Kanerva model for automatic speech recognition. *Computer Speech and Language* 3(1):61–81.

Rogers, D. 1989a. Statistical prediction with Kanerva's sparse distributed memory. In D.S. Touretzky (ed.), *Neural Information Processing Systems* 1:586–593 (Proceedings NIPS–88) (San Mateo, Calif.: Kaufmann).

Rogers, D. 1989b. Kanerva's sparse distributed memory: An associative memory algorithm well-suited to the Connection Machine. *International Journal of High Speed Computing* 1(2):349–365.

Rogers, D. 1990a. Predicting weather using a Genetic Memory: A combination of Kanerva's sparse distributed memory and Holland's genetic algorithms. In D.S. Touretzky (ed.), *Neural Information Processing Systems* 2:455–464 (Proceedings NIPS–89) (San Mateo, Calif.: Kaufmann).

Rogers, D. 1990b. BIRD: A General Interface for Sparse Distributed memory Simulators. Report RIACS TR 90.3, Research Institute for Advanced Computer Science, NASA Ames Research Center.

Rosenblatt, F. *Principles of Neurodynamics.* Washington, D.C.:Spartan.

Rumelhart, D.E., and J.L. McClelland., eds. 1986. *Parallel Distributed Processing,* volumes 1 and 2. Cambridge, Mass.: Bradford/MIT Press.

Saarinen, J., M. Lindell, P. Kotilainen, J. Tomberg, P. Kanerva, and K. Kaski. 1991a. Highly parallel hardware implementation of sparse distributed memory. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (eds.), *Artificial Neural Networks* 1:673–678 (Proceedings ICANN–91) (Amsterdam: Elsevier/North–Holland).

Saarinen, J., S. Pohja, and K. Kaski. 1991b. Self-organization with Kanerva's sparse distributed memory. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (eds.), *Artificial Neural Networks* 1:285–290 (Proceedings ICANN–91) (Amsterdam: Elsevier/North–Holland).

Sejnowski, T.J., and C.R. Rosenberg. 1986. NETtalk: A Parallel Network that Learns to Read Aloud. Report JHU/EECS-86/01, Department of Electrical Engineering and Computer Science, Johns Hopkins University. Reprinted in J.A. Anderson and E. Rosenfeld (eds.), *Neurocomputing: Foundations of Research,* pp. 663–672 (Cambridge, Mass.: MIT Press).

Willshaw, D. 1981. Holography, associative memory, and inductive generalization. In G.E. Hinton and J.A. Anderson (eds.), *Parallel Models of Associative Memory,* pp. 83–104 (Hillsdale, N.J.: Erlbaum).

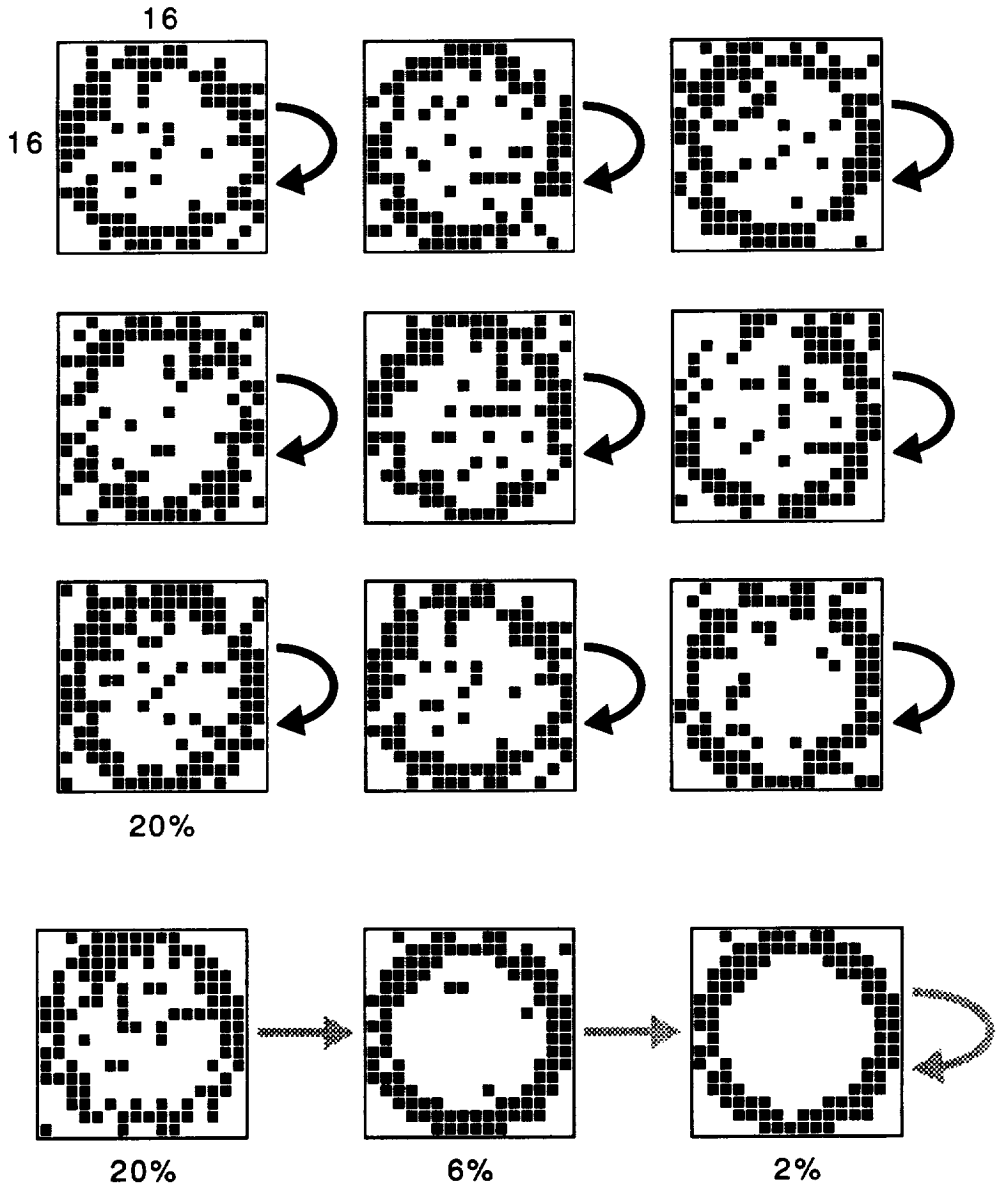

Figure 2–1. Nine noisy words (20% noise) are stored, and the tenth is used as a retrieval cue.

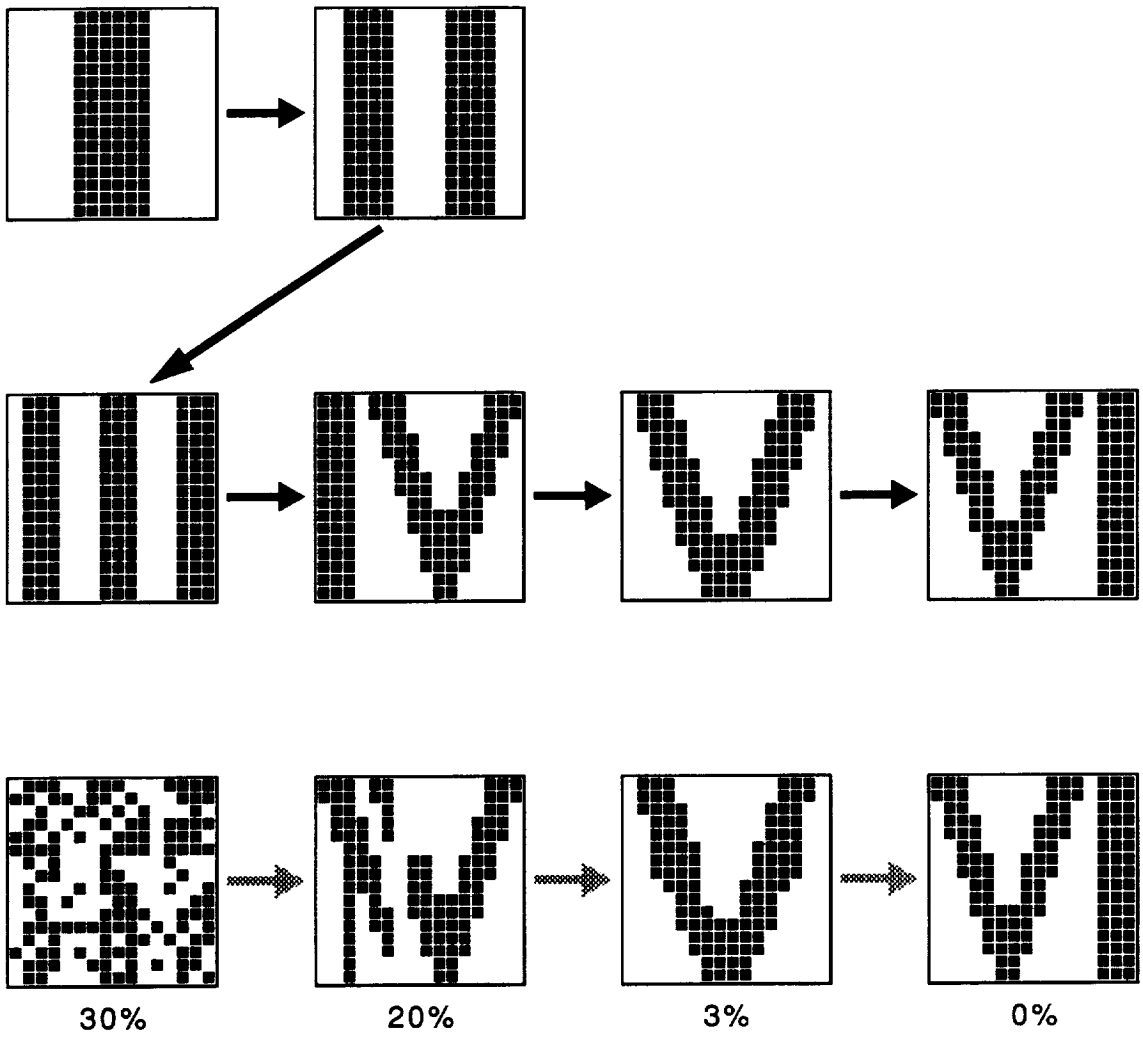

Figure 2–2. Recalling a stored sequence with a noisy (30% noise) retrieval cue.

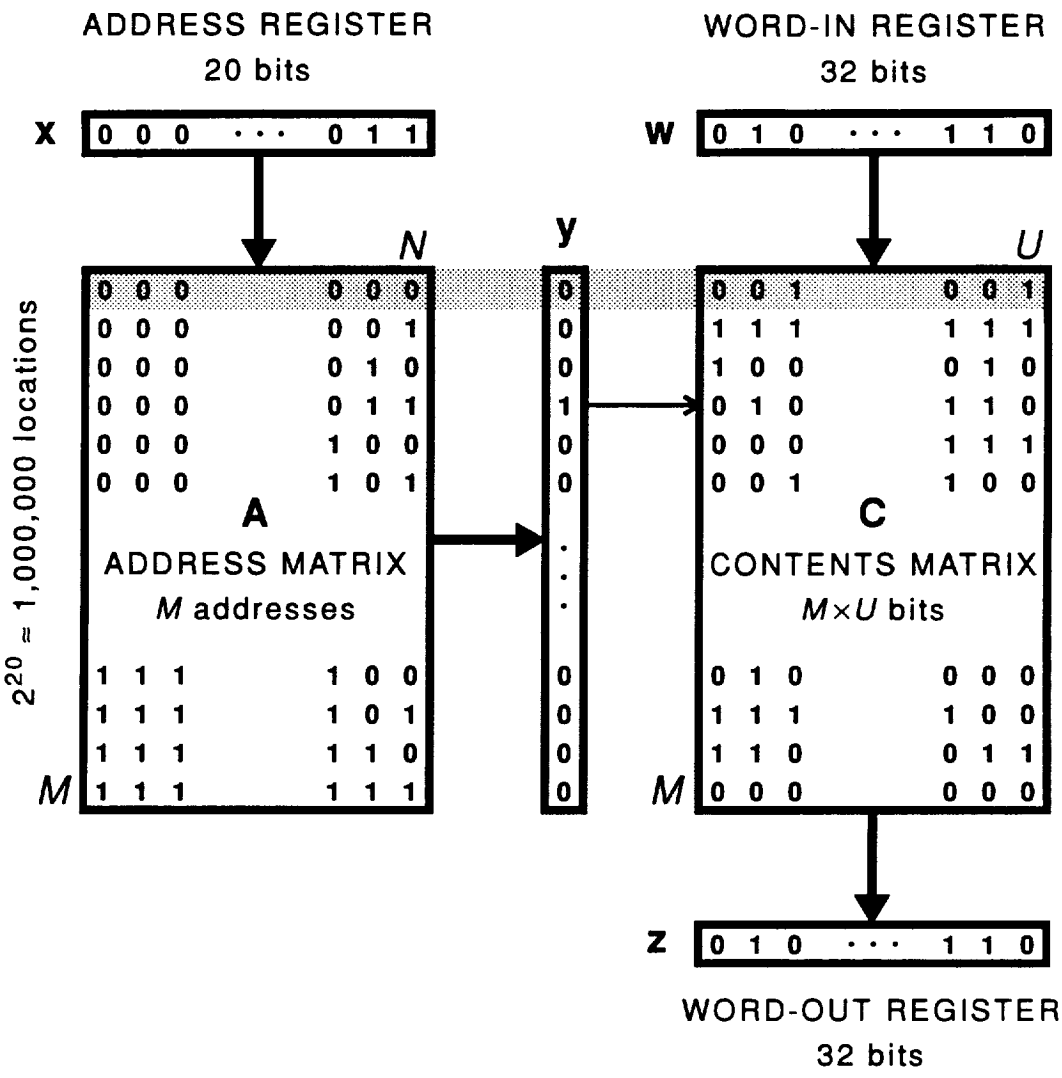

Figure 2–3. Organization of a random-access memory. The first memory location is shown by shading.

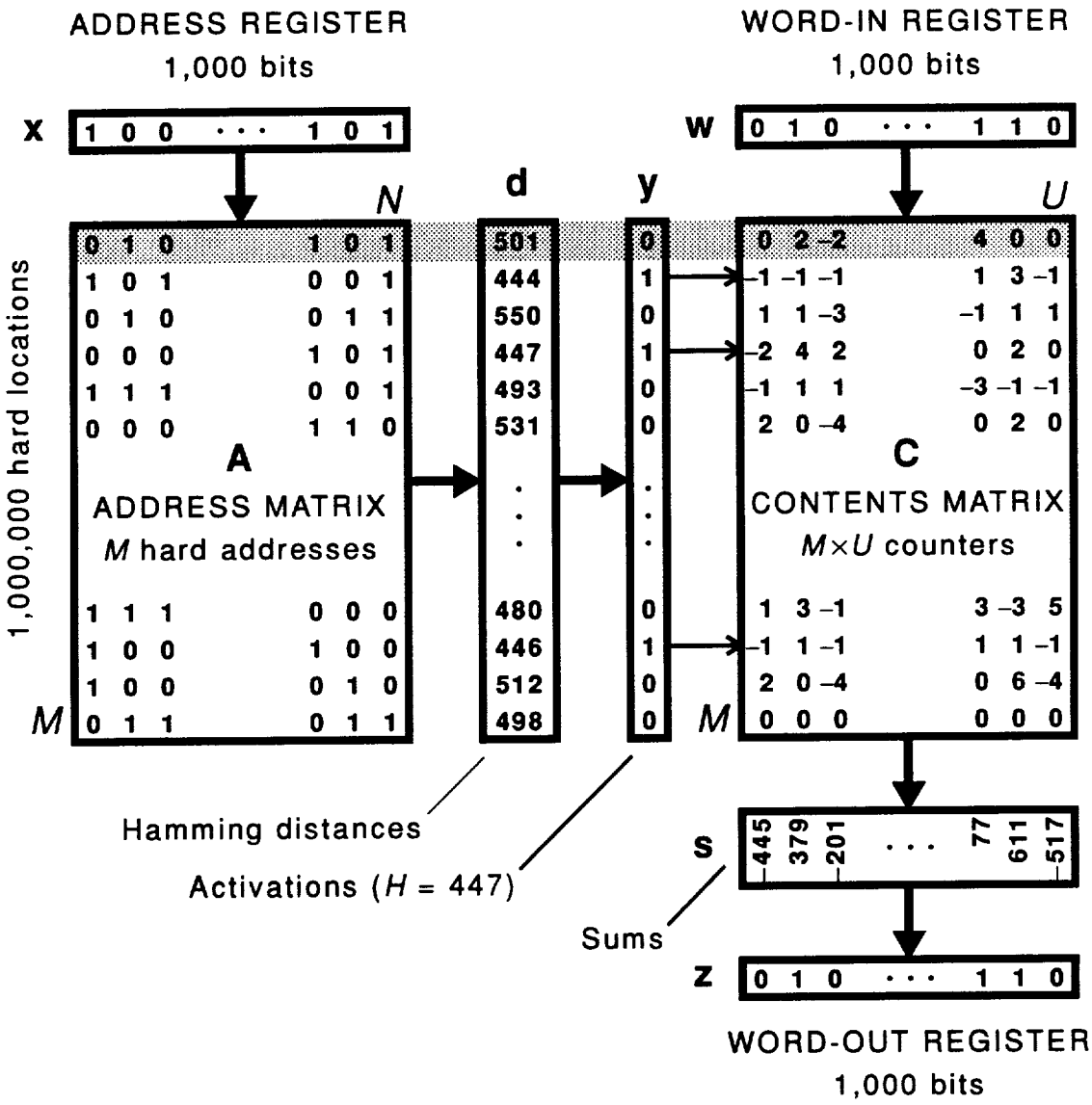

Figure 2–4. Organization of a sparse distributed memory. The first memory location is shown by shading.

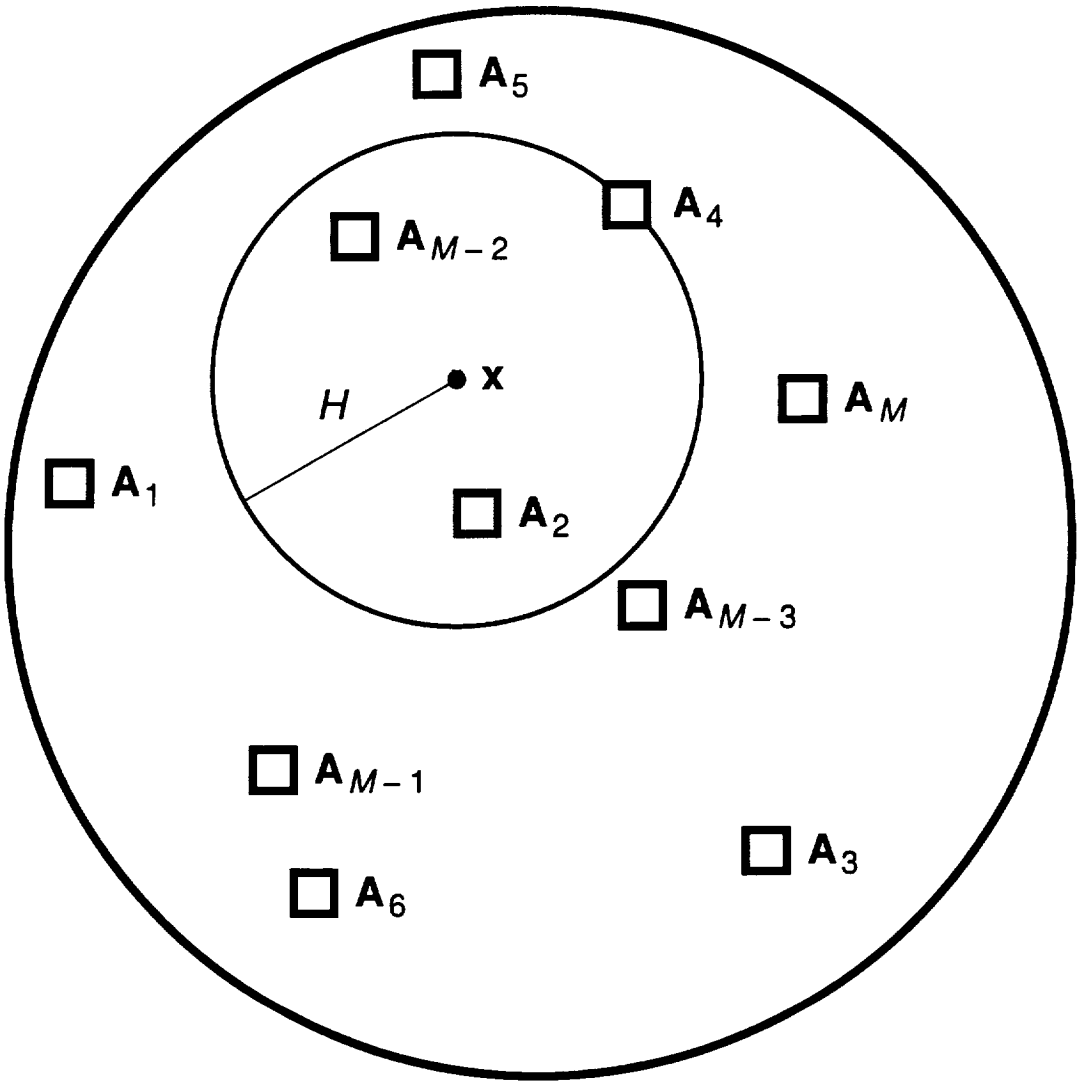

Figure 2–5. Address space, hard locations, and the set activated by **x**. H is the (Hamming) radius of activation.

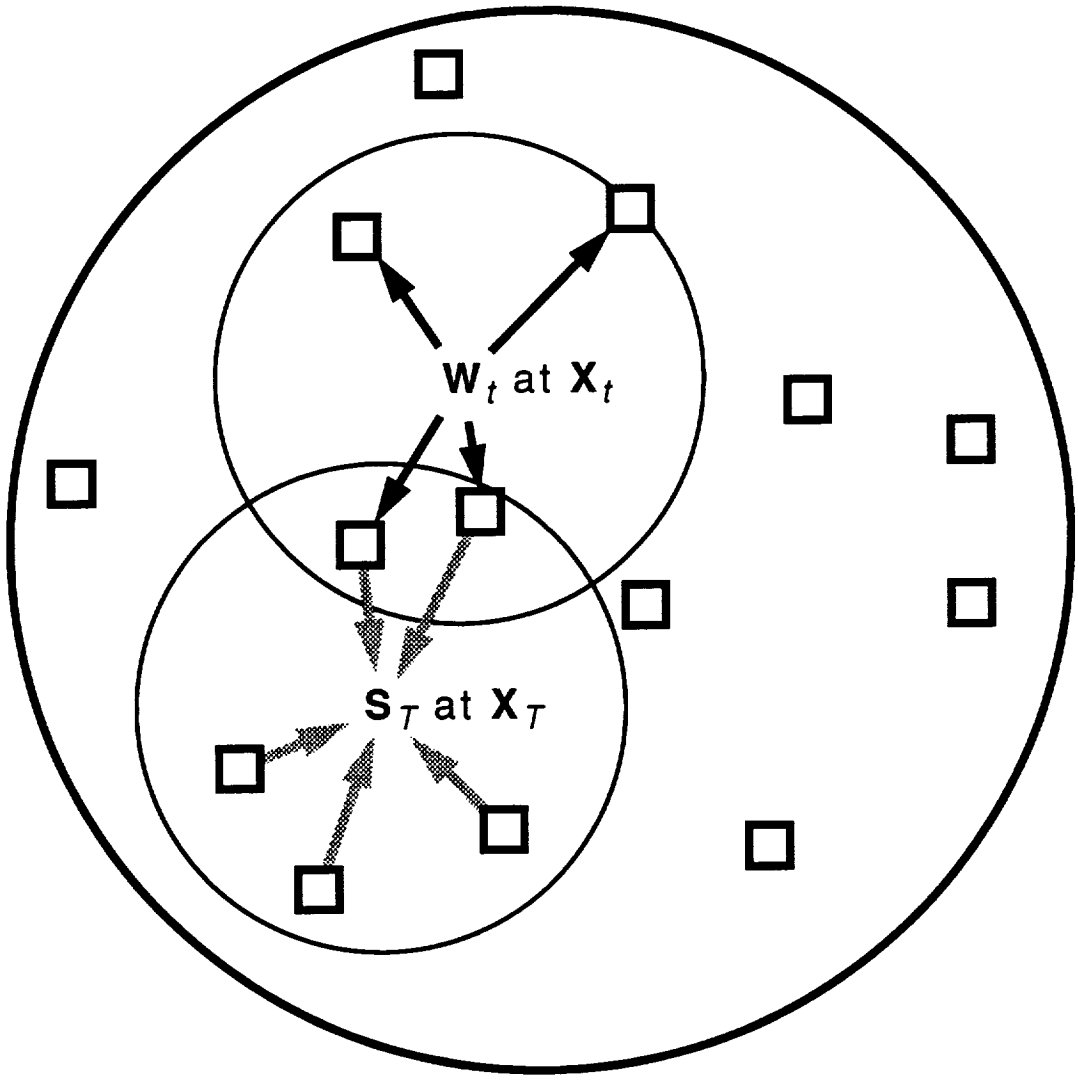

Figure 2–6. Activation overlaps as weights for stored words. When reading at X_T, the sum S_T includes one copy of the word W_t from each hard location in the activation overlap (two copies in the figure).

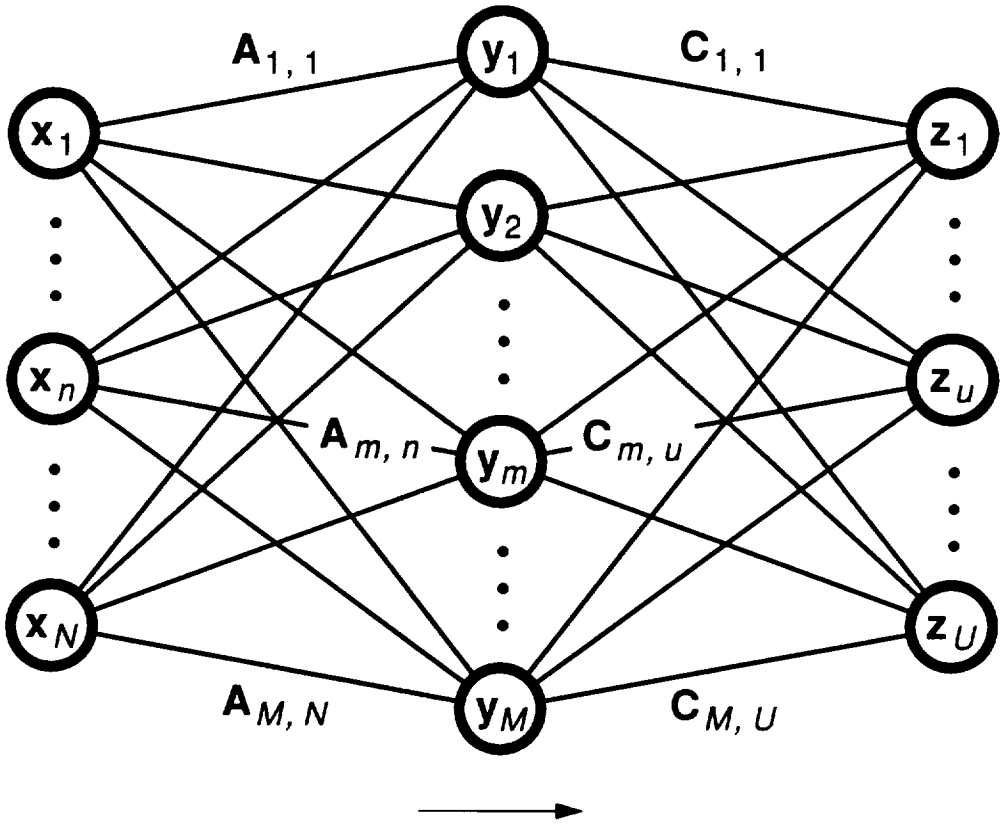

Figure 2–7. Feed-forward artificial neural network.

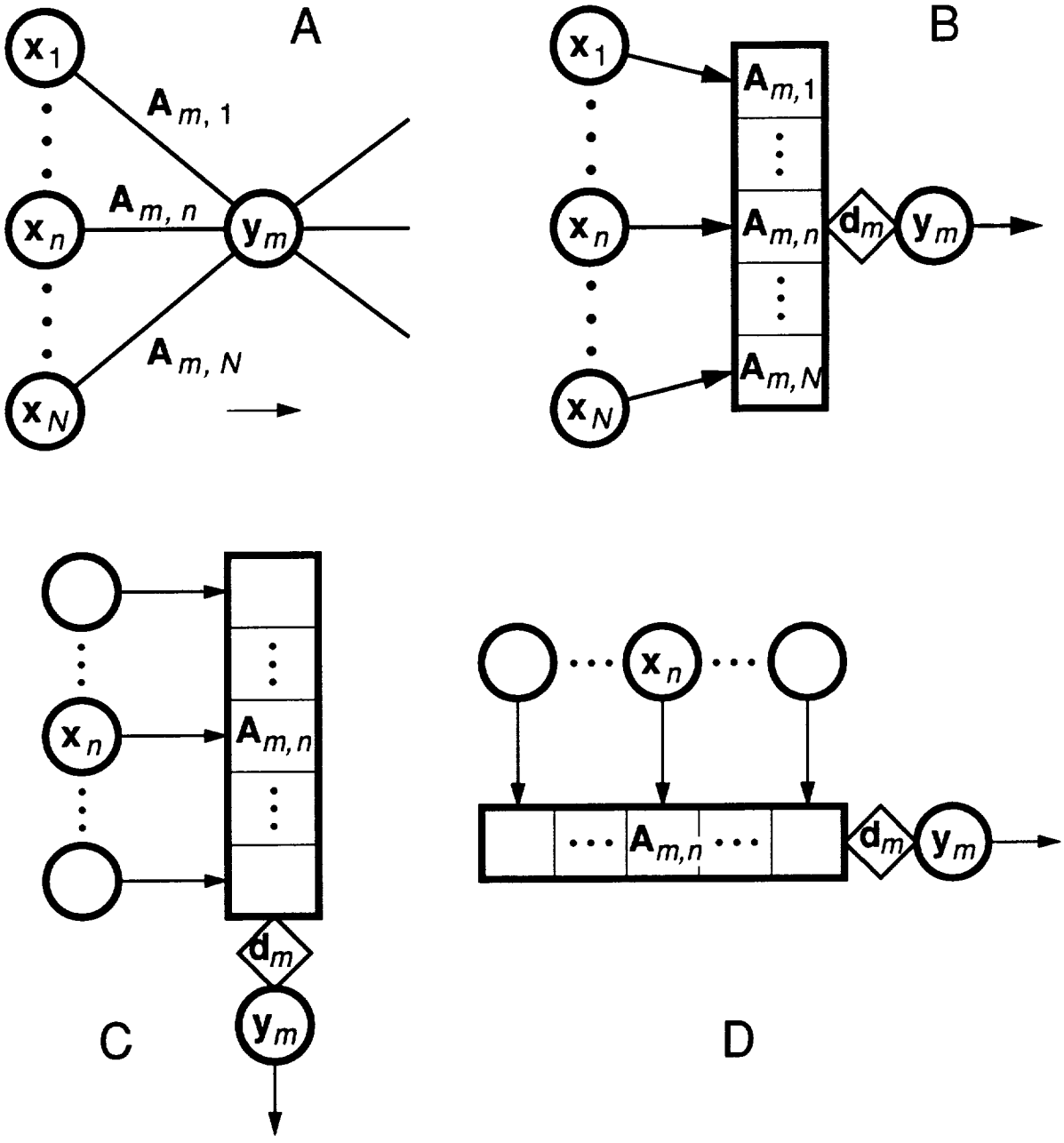

Figure 2–8. Four views of an artificial neuron.

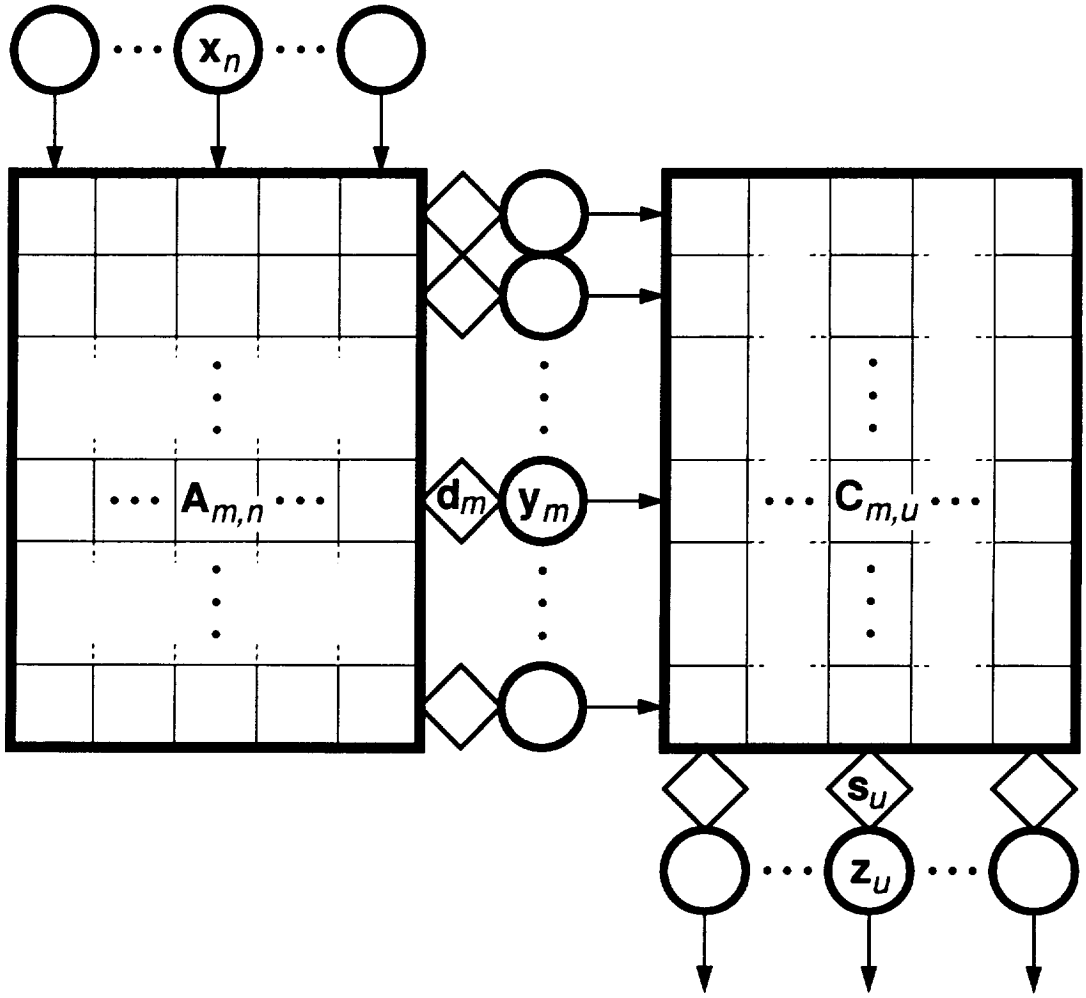

Figure 2–9. Sparse distributed memory as an artificial neural network (Fig. 2–7 redrawn in the style of Fig. 2–4).

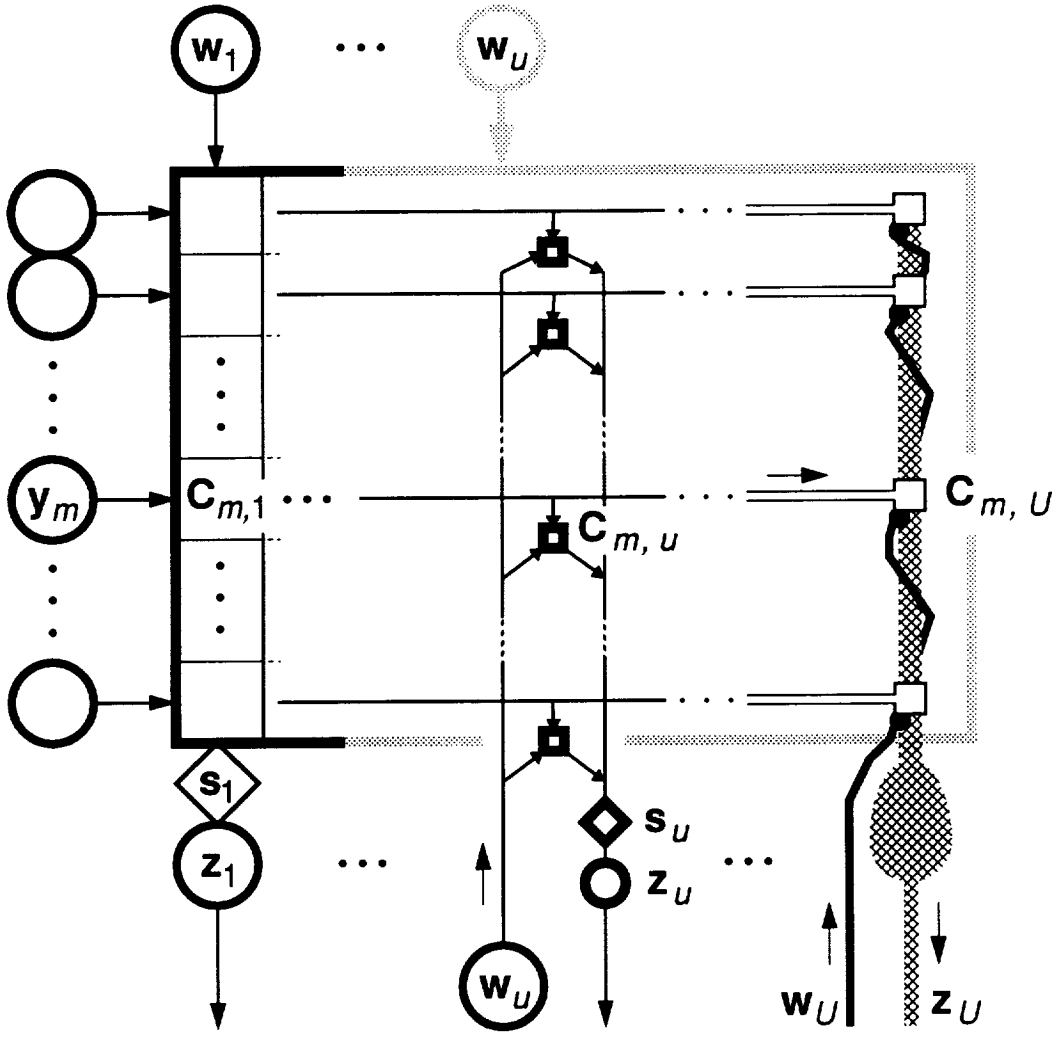

Figure 2–10. Connections to an output neuron. Three output units are shown. The first unit is drawn as a column through the contents matrix C, the middle unit shows the connections explicitly, and the last unit corresponds to Figure 2–11.

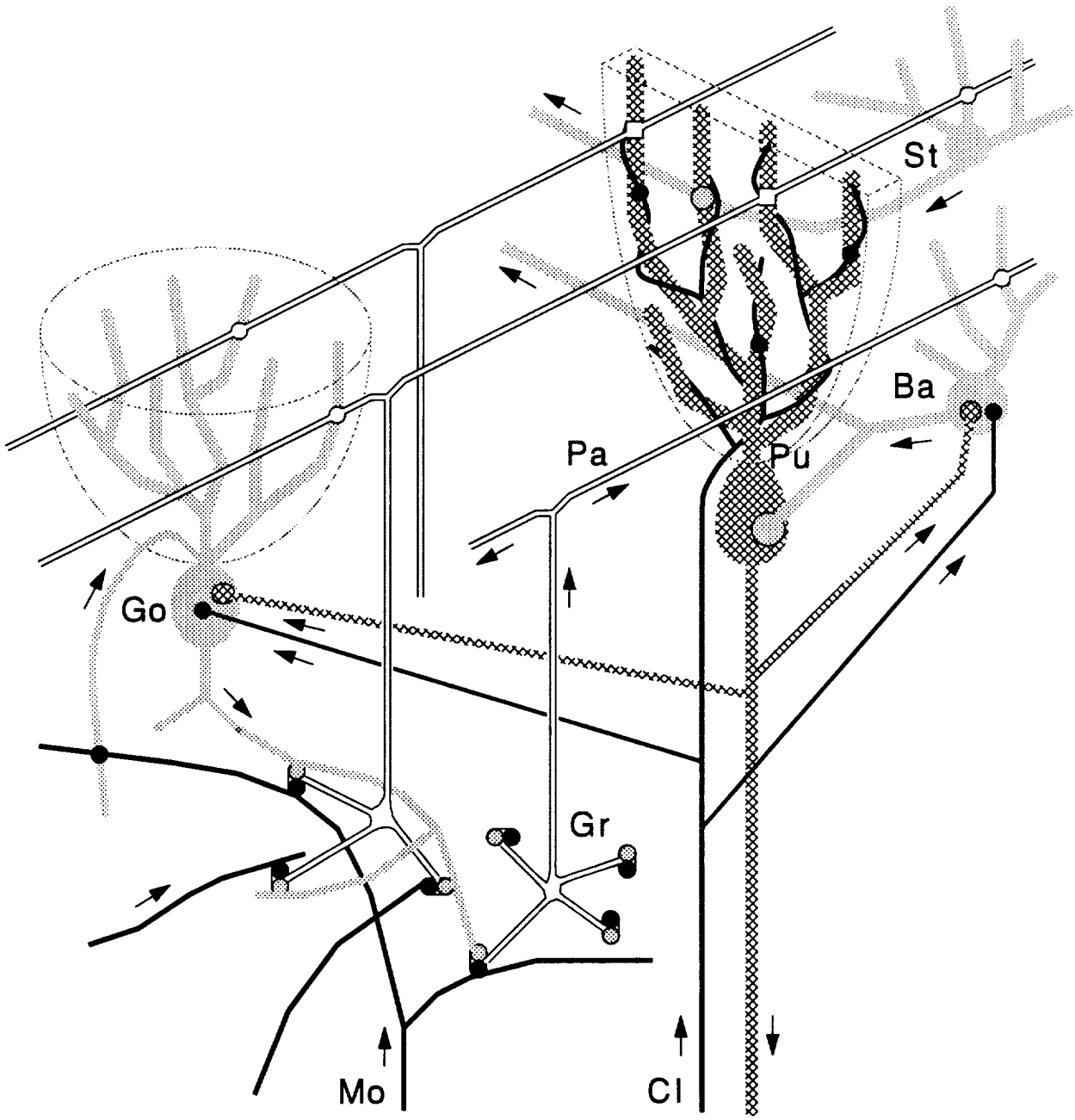

Figure 2–11. Sketch of the cortex of the cerebellum. Ba = basket cell, Cl = climbing fiber (black), Go = Golgi cell, Gr = granule cell, Mo = mossy fiber (black), Pa = Parallel fiber, Pu = Purkinje cell (cross-hatched), St = stellate cell. Synapses are shown with small circles and squares of the axon's "color." Excitatory synapses are black or white, inhibitory synapses are cross-hatched or gray.

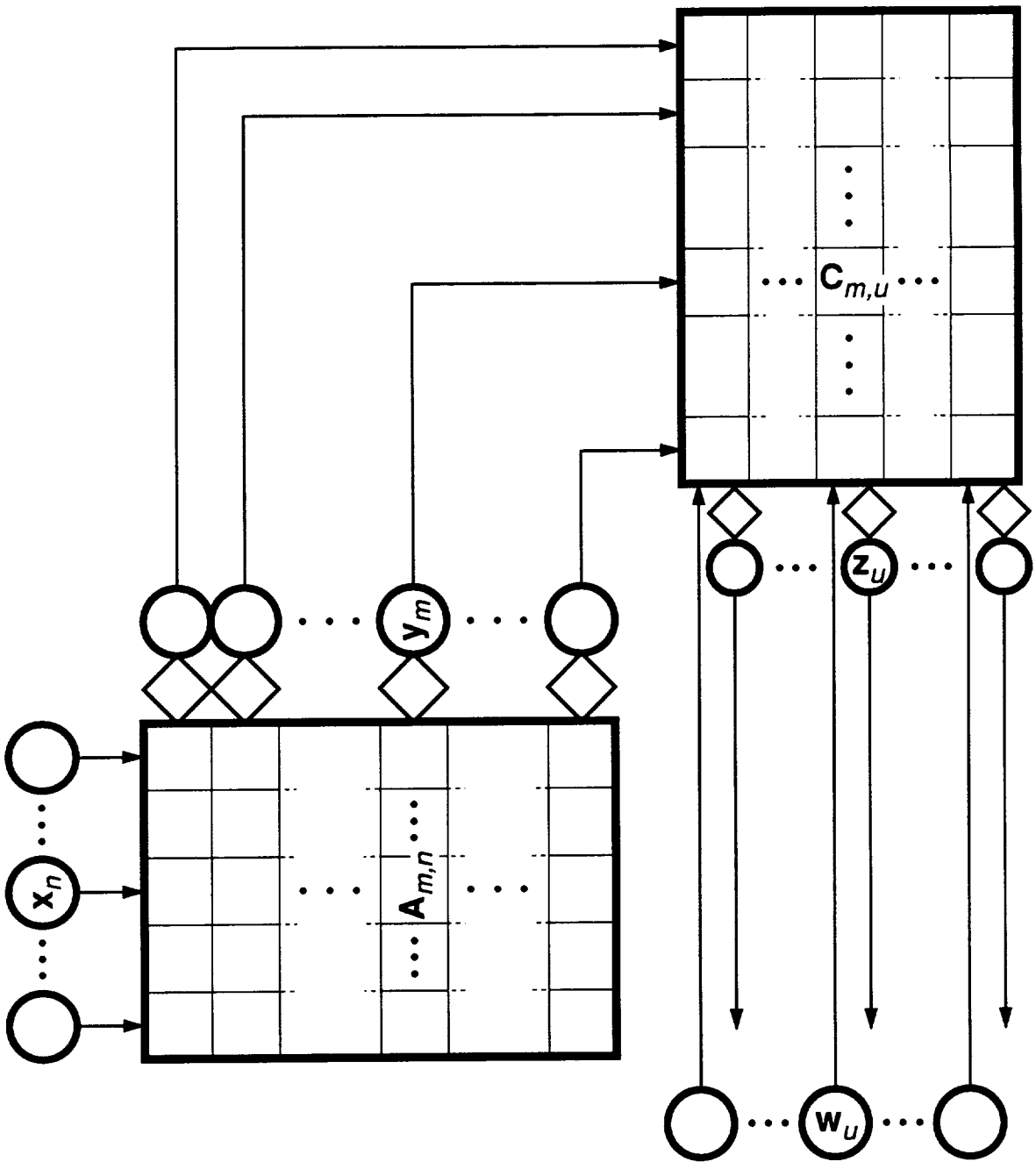

Figure 2–12. Sparse distributed memory's resemblance to the cerebellum (Fig. 2–9 redrawn in the style of Fig. 2–11; see also Fig. 2–10).

Table 2–1

Encoding a 17-level Input Variable s_n in 20 Bits ($K = 4$)

s_n	Input bit																			
	F	M	S	A	G	N	T	B	H	P	V	C	J	Q	W	D	K	R	X	E
0	+	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
1	−	+	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
2	−	−	+	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−
3	−	−	−	+	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	−
4	−	−	−	−	+	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−
5	−	−	−	−	−	+	+	+	+	−	−	−	−	−	−	−	−	−	−	−
6	−	−	−	−	−	−	+	+	+	+	−	−	−	−	−	−	−	−	−	−
7	−	−	−	−	−	−	−	+	+	+	+	−	−	−	−	−	−	−	−	−
8	−	−	−	−	−	−	−	−	+	+	+	+	−	−	−	−	−	−	−	−
9	−	−	−	−	−	−	−	−	−	+	+	+	+	−	−	−	−	−	−	−
10	−	−	−	−	−	−	−	−	−	−	+	+	+	+	−	−	−	−	−	−
11	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+	−	−	−	−	−
12	−	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+	−	−	−	−
13	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+	−	−	−
14	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+	−	−
15	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+	−
16	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+

				A				B				C				D				E
	F				G				H				J				K			
		M				N				P				Q				R		
			S				T				V				W				X	