58205
N92-14695 P7

A CEREBELLAR-MODEL ASSOCIATIVE MEMORY AS A GENERALIZED RANDOM-ACCESS MEMORY

Pentti Kanerva

Research Institute for Advanced Computer Science
NASA Ames Research Center, M/S 230-5
Moffett Field, CA 94035

ABSTRACT

A versatile neural-net model is explained in terms familiar to computer scientists and engineers. It is called the sparse distributed memory, and it is a random-access memory for very long words (for patterns with thousands of bits). Its potential utility is the result of several factors: (1) A large pattern representing an object or a scene or a moment can encode a large amount of information about what it represents. (2) This information can serve as an address to the memory, and it can also serve as data. (3) The memory is noise tolerant--the information need not be exact. (4) The memory can be made arbitrarily large and hence an arbitrary amount of information can be stored in it. (5) The architecture is inherently parallel, allowing large memories to be fast. Such memories can become important components of future computers.

Introduction

This paper deals with neurally motivated associative memory, which is a basic component of neurocomputing. One specific cerebellar-model associative memory is discussed. It is called the sparse distributed memory or SDM [1], and it is described here by comparing it to the ordinary random-access memory (RAM) of a computer. Many of its properties are shared by most neural models, but some are specific to cerebellar models and to the sparse distributed memory in particular. The two cerebellar models that predate the sparse distributed memory and that resemble it the most were developed by David Marr [2] and by James Albus [3, 4].

Description of the Memory

Overview

An ordinary computer memory is a memory for short strings of bits, typically 8, 16, 32, or 64 bits. The bit strings are often thought of as binary numbers or "words," but, in general, they are just small patterns of bits. The memory stores them in addressable locations. The addresses to the memory also are short strings of bits. For example, 20 bits will address a memory with one million locations.

The sparse distributed memory is likewise a memory for strings of bits, except that the strings can be hundreds or thousands of bits long. Because the strings are so long, they are best thought of as large patterns. The addresses to the memory also are long strings of bits, or large patterns. In an important class of these memories, the address and data patterns are of equal size. In the examples in this paper the patterns are rather small; they have 256 bits.

Behavior

The behavior of an ordinary computer memory can be described as follows: If the word W has been written with address A, then W can be read back by addressing the memory with A, and we say that A points to W. The condition for this is, of course, that no other word has been written with address A in the meantime.

The sparse distributed memory has like behavior: If the pattern W has been written with pattern A as the address, then W can be read back by addressing the memory with A, and we say that A points to W. However, the conditions for this are more restrictive than they are with ordinary computer memories, namely, that no other pattern has been written before or since with address A or with an address that is similar to A.

The added restrictions pay off in noise tolerance in two ways: To read the pattern W from a sparse distributed memory, the address pattern need not be exactly A (in ordinary RAM, the exact address A must be used to read W). This means that the memory can tolerate a noisy reference address; it can respond to a partial or incomplete cue. Tolerance for noisy data shows up as follows: If many noisy versions of the same target pattern have been written into the memory, a (nearly) noiseless target pattern can be read back.

Figure 1 illustrates the memory's tolerance for noise. This memory works with 256-bit patterns. For ease of comparing patterns with each other, they are displayed on a 16 x 16 grid with 1-bits shown in black. The nine patterns in the upper part of the figure were gotten by taking a circular pattern and changing 20 percent of the bits at random. Each of the patterns was written into the memory with itself as the address. The noisy tenth pattern was then used as the address for reading from the memory, and the relatively noise-free eleventh pattern was retrieved. When that pattern was used as the next read address, the final, nearly noise-free pattern was retrieved. Worth special notice is that the noise-free circular pattern was never used as the write address nor was it ever written into the memory (i.e., the memory had never "seen" the ideal pattern; it created it from the noisy versions it had seen).

The method of storage in which each pattern is written into memory with itself as the address, as illustrated in Figure 1, is called autoassociative. With autoassociative storage, the memory behaves like a content-addressable memory in the following sense: It allows a stored pattern to be retrieved if enough of its components are known.

A more general method of storage in which an address pattern and the associated data pattern are different is called heteroassociative. Figure 2 illustrates its use in storing a sequence of patterns. The sequence is stored as a pointer chain, with the first pattern pointing to the second, the second to the third, and so forth. Any pattern in the sequence can then be used to read out the rest of the sequence simply by following the pointer chain. Furthermore, the cue for retrieving the sequence can be noisy, as shown in Figure 3, in which a noisy third pattern retrieves a less noisy fourth, which in turn retrieves an almost noiseless fifth pattern, and the sixth pattern retrieved is perfect. If the memory's address and data patterns are of different size, only heteroassociative storage is possible, although it is not possible to store pattern sequences as pointer chains.

The term 'associative memory' refers in neurocomputing to this very general property of linking one pattern to another, or forming an association, the linkage being the association. In that broad sense, even the ordinary random-access memory is associative. However, the term is more specific in computer-engineering usage and is usually synonymous with 'content-addressable memory', which, in turn, is a tighter concept in computer engineering than in neurocomputing or psychology. As a neurocomputing term, associative memory implies also noise tolerance as illustrated in the examples above.

Construction

The ordinary computer memory is an array of addressable registers or memory locations. The locations are numbered sequentially, and the sequence number is the location's address. A memory with a thousand locations will therefore need ten-bit addresses. If the memory is built for eight-bit words, each location will have eight one-bit storage bins or flip-flops. This organization of the memory is shown in Figure 4. Each row in the figure is one memory location, with its address shown on the left and the storage bins on the right. In this figure, the memory's contents (the storage bins) have been set at random.

The sparse distributed memory also is an

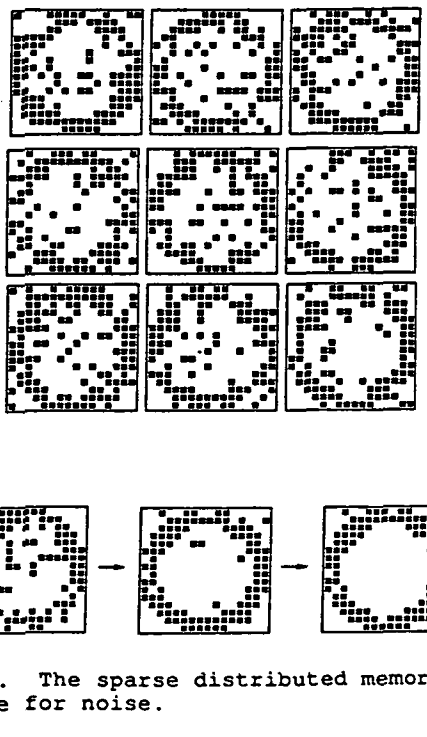
FIGURE 1. The sparse distributed memory's tolerance for noise.

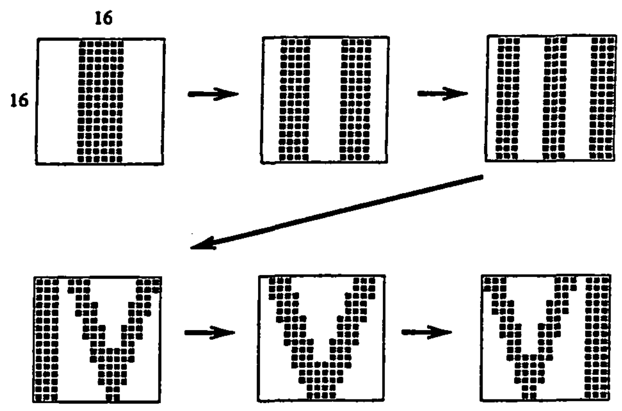

FIGURE 2. A sequence of patterns that is stored as a pointer chain.

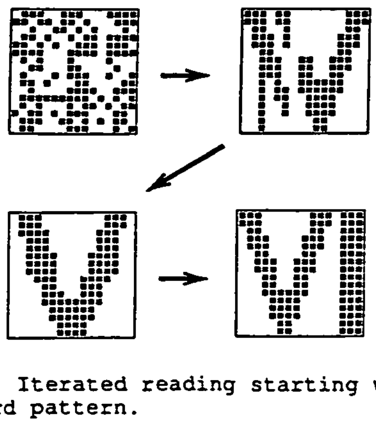
FIGURE 3. Iterated reading starting with a noisy third pattern.

array of addressable registers or memory locations. The addresses of the locations, however, are not sequence numbers but large bit patterns (256-bit addresses in the examples above). To store 256-bit data patterns, each location will have 256 storage bins for small up-down counters. This organization is shown in Figure 5, which is not that different from Figure 4. In Figure 5, the location addresses are random 256-bit patterns, and the memory's contents are shown after many patterns have been stored in the memory.

Because the memory addresses are large bit patterns, the number of addresses and hence of possible memory locations is astronomical. Only memories with a small subset of the possible locations can be built in practice, and that is why these memories are called sparse. Practical numbers range in the thousands to millions to billions of locations.

To move data into and out of the memory array, both kinds of memories have three special (input/output) registers: one for the memory address or cue, another for the word or

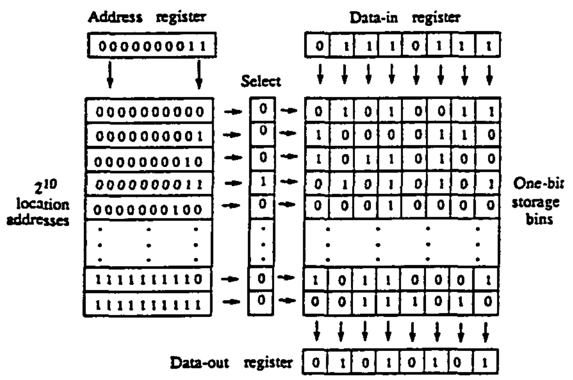

FIGURE 4. The organization of a random-access memory as an array of addressable locations.

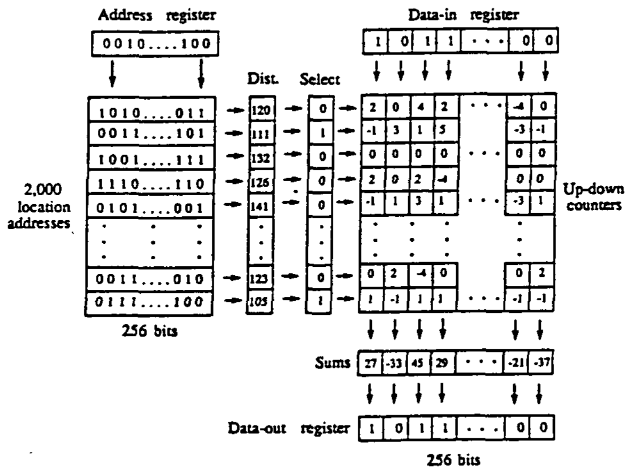

FIGURE 5. The organization of a sparse distributed memory as an array of addressable locations.

data pattern to be written into the memory, and the third for the word or data pattern being read out of the memory. In Figures 4 and 5 they are above and below the memory array.

In addition to the memory array and the input and output registers, Figures 4 and 5 show intermediate results of a memory operation. The numbers in the column or columns between the address matrix (on the left) and the contents matrix (on the right) indicate whether a memory location is selected for a given read or write operation. The selection depends on the contents of the address register, on the location's address, and on the selection criterion, as will be explained shortly. Figure 5 (of SDM) has, in addition, a row of sums as a way of getting from the contents of the memory locations to the final output pattern.

Operation

Reading and writing in ordinary computer memory is simple in concept. Both operations start with specifying a memory address in the address register. That selects one location from the memory array--the location with the matching address. The selection is indicated by the single 1 in the select column of Figure 4, the rest of the values in that column being zeros. If the memory operation is a write, the word being written is placed in the data-in register, and it will replace the word stored previously in the selected location; if it is a read, the contents of the selected location are copied into the data-out register.

Reading and writing in the sparse distributed memory likewise start with addressing the memory. However, when an address is specified in the memory-address register, the memory array will usually not have a location with that exact address. This is overcome by selecting many locations at once--and by modifying the rules for writing and reading accordingly.

The criterion for selecting or activating a location is similarity of address patterns: If the location's address is sufficiently similar to the address in the address register, the location is selected. Hamming distance between address patterns provides a simple measure of similarity, and it is used in Figure 5 and in subsequent examples. The column next to the address matrix in Figure 5 shows these Hamming distances, and the column next to it has ones where this distance does not exceed 112 bits. These than are the selected (nearby, active) locations, the unselected (distant, inactive) locations being indicated by zeros in the select column. As a rule of thumb, the selection criterion should be such that many locations are active at once, but their number should not exceed the square root of the total number of memory locations.

A (data) pattern is written from the data-in register into the memory by adding it into all selected locations (in an ordinary RAM, new data replace old in one location). It can be added simply by incrementing the counters under the ones of the data-in register and by decrementing the counters under the zeros.

Figure 6 shows the writing of two patterns into a very small memory that is initially empty (all counters initially zeros). The selected locations are shown in white and the unselected in gray. As more and more data are written into the memory, individual counters can reach their capacity. When this happens, attempts to increment a counter past its maximum value or to decrement it past its minimum value are ignored.

A pattern is read out of the memory (from the selected locations) by computing an average over the contents of the selected locations. A simple average is gotten by adding the contents (vector addition) and by thresholding the sums at zero, with a sum larger than zero yielding a 1 in the output pattern, and a sum smaller than or equal to zero yielding a 0. A bit of the output pattern will then be 1 if, and only if, the patterns written into the currently active locations have more ones than zeros in that bit position, constituting a bitwise majority rule. Figures 7a and 7b illustrate reading at and reading near the second write address, respectively. In both cases, the second written pattern is retrieved (cf. Fig. 6b).

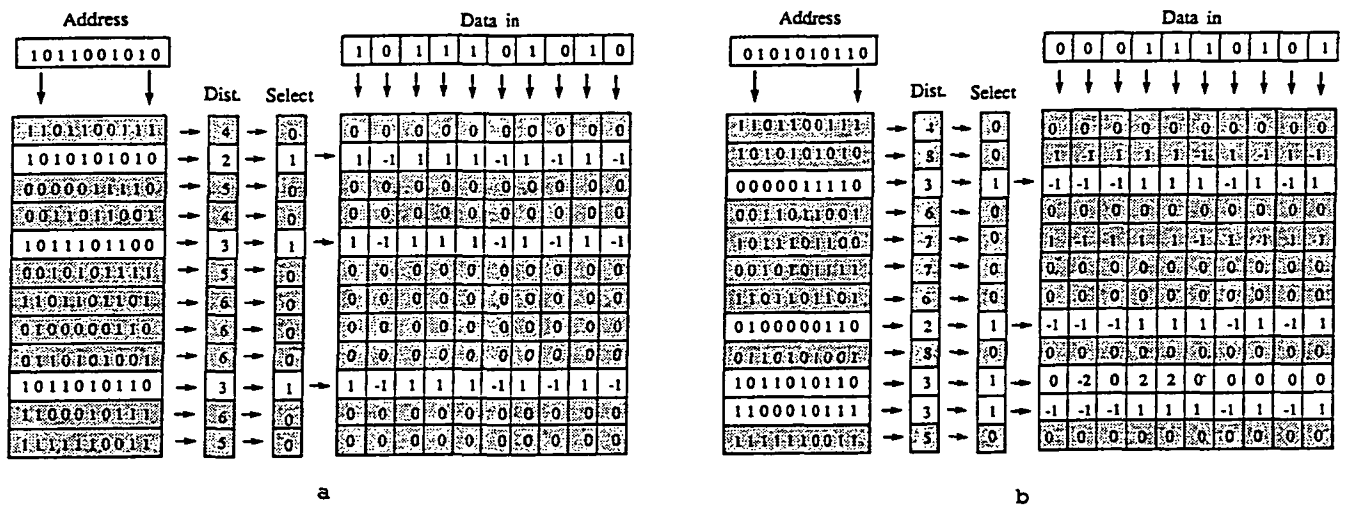

FIGURE 6. Writing two patterns into a tiny sparse distributed memory. First the pattern 1011101010 is written at 1011001010 (a), and then the pattern 0001110101 at 0101010110 (b).

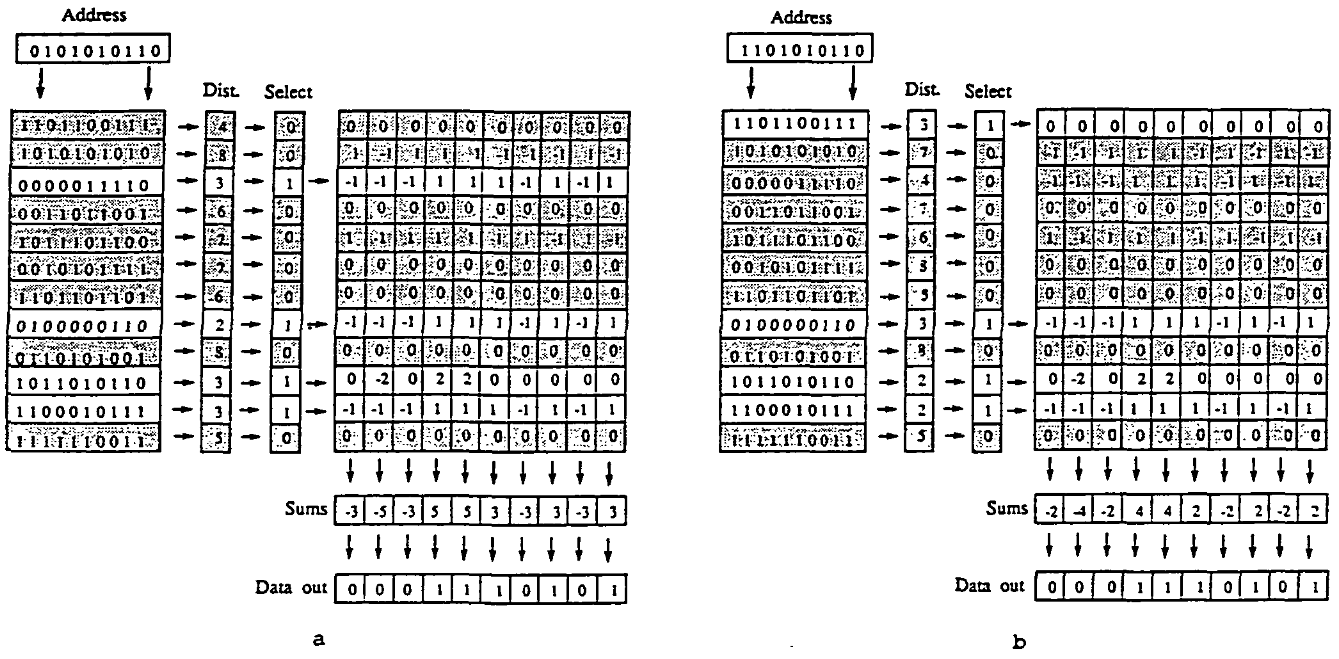

FIGURE 7. Reading at (a) and reading near (b) a previous write address.

Why Does the SDM Work?

A premier property of the sparse distributed memory is sensitivity to similarity, or noise tolerance. It is the result of distributing the data, that is, of writing into and reading from many locations at once, and it is explained mathematically by the amount of overlap, counted in active memory locations, when the memory is addressed with two different patterns. If two address patterns are very similar to each other, the sets of locations they activate have many locations in common; if they are dissimilar, the common locations are few or none. This can be seen in Figures 6 and 7: The second read address (Fig. 7b) differs from the second write address (Fig. 6b) by one bit only (the two addresses are very similar), and the number of locations selected by both--the overlap--is 3; it differs from the first write address (Fig. 6a) by five bits (dissimilar), and the overlap is 1 location. Thus, when we read near the second write address (Fig. 7b), the second written data pattern has a weight 3 and the first a weight 1 in the sums accumulated from the selected locations, allowing the second pattern to be recovered in thresholding.

The example illustrates that, in a sparse distributed memory, common address bits translate into common memory locations, and common memory locations translate into weights for stored patterns when reading from the memory. Thus, the memory is a means of realizing a weighting function that gives low weights to most of the patterns written into the memory and high weights only to a small number of "relevant" patterns, the relevance being judged by similarity of address.

The operation of the memory is statistical, and the actual output is affected not only by the construction of the memory but also by the structure of the data. The results discussed above are demonstrated most readily when the addresses of the locations and the data are a uniform random sample of their respective spaces of bit strings. There is the further condition that not too many patterns have been written into the memory. The memory works in the manner described if the number of stored patterns is no more than 1-5 percent of the number of memory locations.

Closely Related Architectures

Ordinary RAM as a Special Case of SDM

We can now demonstrate the close kinship of the two kinds of memories. Let us start with a random-access memory that has just over 16 million (2 to power 24, 2**24) locations for 32-bit words. The memory address is then 24 bits long. This memory can be thought of as a sparse distributed memory with the following parameters: an array of 2**24 memory locations, with 24-bit addresses and 32 one-bit up-down counters for holding the data. The address matrix would contain each of the 2**24 possible addresses exactly once, and the Hamming distance for selecting a location would be zero. That would mean that each possible address would select exactly one location, and two different addresses would always select two different locations.

Writing into this memory causes the old contents of the location to be lost to over- and underflow, because the location's counters have only one bit each. Reading from it fetches the contents of one location--whatever was written there last--and thresholding will not change the bits. This example shows that the sparse distributed memory indeed is a generalized random-access memory; it yields the ordinary RAM as a special case. In the terminology of the preceding section, the data pattern associated with the read address has weight one and all other patterns have weight zero.

Extensions of the Basic Model

In the basic model of sparse distributed memory, the pattern components are binary. The model can be generalized to allow many-valued components, including continuous, and an important case is one in which the components are trinary. The most convenient three values are -1, 0, and 1, and useful interpretations for them are 'off', 'don't know or don't care', and 'on', respectively. The activation of a location must be based on a measure that is more general than the Hamming distance, for example, on the inner (dot) product of the location's address with the address in the address register. Writing into the memory is by adding the input-data pattern into the active locations, much as before, and reading is by summing over the active locations, except that to get the final output pattern, we need two thresholds instead of one. If this model is restricted to the values -1 and 1, and the two thresholds are both equal to 0, it is equivalent to the basic model with binary components.

Other variations of the model are gotten by adjusting it to the data being stored. The more the data deviate from the "ideal," that is, from being a uniform random sample of the underlying space, the more important the adjustments are. Real-world data are never ideal in that sense, and so the adjustments are essential in systems for real-world applications. The adjustments include: choosing the addresses of the memory locations based on the addresses in the data; activating a fixed number of closest locations in any given read or write operation instead of all locations within a certain distance; having individual selection distances for individual locations; adding correction vectors into the memory instead of, or in addition to, data-pattern vectors; weighting active locations in a read operation according to their contents; and adjusting the thresholds that determine the final output.

Some variations of the basic model would take it outside the realm of cerebellar models. Adjusting the addresses of the memory locations as a part of "training" the memory for a given data set is the most important of such variations. In the cerebellar models, the address of a location, once defined, stays fixed, setting them apart from more general models, such as multilayer back-propagation nets [5], which resemble the cerebellar models in many other respects. Another characteristic of the cerebellar models, as compared with most other models, is that any given read or write operation activates many locations but leaves most locations inactive: a location is either on of off, as indicated in the select column of Figure 5. These constraints of the cerebellar models simplify the construction of memories based on the models, making it possible to build very large memories that can be trained reasonably fast.

In the taxonomy of adaptive networks or artificial neural nets, the sparse distributed memory is a 'fully connected three-layer feed-forward' net. The address register (see Fig. 5) corresponds to the input layer of such a net, the location-address matrix holds the input weights of the hidden layer (each memory location--a row--is one hidden unit), the select vector is the output of the hidden layer, the contents matrix (the up-down counters) are the weights of the output layer (each column is one output unit), and the data-out register has the outputs of the output layer. 'Fully connected' means that each bit of the input address is seen by each memory location and that each memory location can contribute to each output bit. 'Feed forward' means that the output of one layer goes to the next or subsequent layers only (no direct feedback to the layer itself or to its predecessors), which in turn means that the outputs of a layer are logically independent of each other. The term 'three-layer' is a misnomer, as is evident when several such nets are cascaded or pipelined. Cascading three of them will not result in a nine-layer net but in a seven-layer net, which suggests that the original net really is a two-layer net (and a cascade of three of which is a six-layer net). Thus, the network input (the address register) should not be counted as a separate layer.

Relation to the Cerebellum

The reason for calling the sparse distributed memory, and the models of Marr and of Albus, cerebellar models is largely historical. After developing these neural models of associative memory, the developers noticed and pointed out remarkable similarities in the wiring diagrams of their models and the wiring of the cortex of the cerebellum and, based on the similarities, suggested functions for several cell types of the cortex. The significance of the models is in giving us a mathematical way to look at a major part of the brain, in the perspective of the cerebellum as an associative memory with billions of locations, in motivating further research into the cerebellum, and in arming researchers with useful questions.

Why Associative Memories?

Nature has solved problems that appear to be beyond the capacity of even the most powerful computers. These problems include taking a complex signal from the world, such as the raw input to our visual, auditory, olfactory, and tactile systems, and producing from it over time a coherent model of the world--and of the self in it--that allows us to function in the world. In our ability to do so, we think of ourselves as intelligent and would call systems with similar powers intelligent. How do intelligent systems work? We will consider this question only as it relates to associative memories for large patterns.

The perceptual task of identifying an incoming signal based on experience can be divided into sensory analysis and pattern matching. In sensory analysis, the senses extract features from the signal, and further processing of the signal is in terms of those features. If two scenes produce very similar patterns in terms of the extracted features, the two will be identified as the same by an associative memory (cf. Fig. 1). This is exactly what an intelligent system has to do in identifying objects from different views of it. However, it is important that the features are appropriate for the task. For a counter-example, the pixels of a bit map (a raw retinal image) are poor features for vision, because shifting the figure only slightly or viewing it from a different distance can change a large portion of the features. Human and animal perceptual systems have attentional mechanisms, including feedback from memory, that help the sensors to extract appropriate features.

The actions of humans and animals are accomplished by the selective contraction and relaxation of large numbers of muscle fibers controlled by large numbers of motor neurons. The configuration of active and inactive motor neurons at any one time is a large pattern, and the state of no single neuron is critical for the performance of a given action. The activation patterns of motor neurons are therefore appropriate for an associative memory, as is the learning of actions as responses to sensory patterns--the actions being associated with the sensations. Actions can also be associated with internal states of the system that reflect the system's past in complicated ways, which means that a system based on an associative memory can learn complex, coordinated actions.

The relative merits of associative memories in these tasks derive from how information is packaged. Conventional computers work with small bit patterns (words) that represent a quantity, an index, or a small vector of features. Many such patterns are needed to describe a complex object or a moment of experience. However, at the top level, a single, short index describes or encodes it. The top-level description is precise, as two slightly different indexes can point to two entirely different objects, but it is also almost totally uninformative. To find out anything about the object, it is necessary to fetch from memory further indexes and associated data fields. This allows objects to be described in arbitrary detail, but it also tends to hinder fundamental operations such as the comparison of objects to see how they are related--it makes "seeing" objects in whole difficult; they are seen in tiny fragments.

In contrast, systems based on neurally motivated associative memories work with large patterns (e.g., 10,000 bits) as units. A single pattern can encode a large amount of information about an object--hence it is highly informative, yet it need not be precise. It can serve as the (top-level) description of an object, and it can also serve as an index.

These properties of the descriptions, together with the properties of the memory, are helpful with operations such as the comparing of objects. They also make it easy to describe events that occur over time: a moment (of experience) can be encoded by a single pattern, and an event by a sequence of patterns that is stored in the memory as a pointer chain. A single pattern can include sensory and motor components, plus components that encode the internal (subjective) state of the system, and hence a sequence of patterns can encode interactions of all of these components.

The memory's ability to store associations, and pattern sequences in particular, gives it the power to predict, and the failure of a prediction signals an occasion for learning. Learning is by training through a set of examples rather than by explicit programming. This is referred to as learning from experience. The term is particularly appropriate if the training patterns encode real-world phenomena.

Among traditional methods, multivariate statistical analysis resembles associative-memory-based methods, and there are important connections to coding theory and to adaptive filters. All of these exploit the richness of the geometry of very-high-dimensional spaces, something that conventional computer methods tend not to do.

Pattern Computing

Neurally motivated associative memories and, more generally, adaptive networks or artificial neural nets are computing architectures for very large patterns. They are therefore classified appropriately as pattern computers, as contrasted with conventional numeric and symbolic computers. This classification is based on practical considerations, as a computer in any one class can be used to emulate those in the other two, except that the emulations tend to be too slow to be of practical interest. The speed of pattern computers in dealing with very large patterns is achieved by large numbers of relatively simple processors working in parallel.

Today's computers combine components for numeric and symbolic computing. We can expect future computers to add more and more pattern-computer components to them, as we learn to build and use pattern computers. That, in turn, will broaden the scope of computing and the usefulness of computers--it may well revolutionize computing.

Acknowledgements. This work was supported by the National Aeronautics and Space Administration (NASA) cooperative agreement NCC2-408 with the Universities Space Research Association (USRA). Michael Raugh fended off interruptions and made it possible to write the paper. The examples were run on an SDM simulator written for the SUN workstation by Robert Brown. Louis Jaeckel came up with the idea for Figure 1. James Keeler first drew the matrix pictures of the SDM as they appear in this paper, and he called the zeros in the trinary model don't-know-or-don't-care bits. David Rogers prepared the illustrations. The author's sincere thanks to all.

References

[1] P. Kanerva, Sparse distributed memory, MIT Press, 1988.

[2] D. Marr, "A theory of cerebellar cortex," J. Physiology, vol. 202, p. 437, 1969.

[3] J. S. Albus, "A theory of cerebellar functions," Mathematical Biosciences, vol. 10(1/2), p. 25, 1971.

[4] J. S. Albus, Brains, behavior, and robotics, Byte Books, 1981.

[5] D. E. Rumelhart and J. L. McClelland, eds., Parallel distributed processing, vol. 1, MIT Press, 1986.