Extended Sparse Distributed Memory and Sequence Storage

Javier SNAIDER[1], Stan FRANKLIN[2]
Computer Science Department & Institute for Intelligent Systems, The University of Memphis
[1]*FedEx Institute of Technology #403h, 365 Innovation Dr., Memphis, TN 38152; E-mail: jsnaider@memphis.edu*
[2]*FedEx Institute of Technology #312, 365 Innovation Dr., Memphis, TN 38152*

Abstract. Sparse distributed memory (SDM) is an auto-associative memory system that stores high dimensional Boolean vectors. SDM uses the same vector for the data (word) and the location where it is stored (address). Here we present an extension of the original SDM that uses word vectors of larger size than address vectors. This extension preserves many of the desirable properties of the original SDM: auto-associability, content addressability, distributed storage, robustness over noisy inputs. In addition, it adds new functionality, enabling an efficient auto-associative storage of sequences of vectors, as well as of other data structures such as trees. Simulations testing this new memory are described.

Keywords. Sparse Distributed Memory, Episodic Memory, Sequence Representation, Cognitive Modeling

Introduction

First proposed by Kanerva [1], sparse distributed memory (SDM) is based on large binary vectors, and has several desirable properties. It is distributed, auto associative, content addressable, and noise robust. Moreover, this memory exhibits interesting psychological characteristics as well (interference, knowing when it doesn't know, the tip of the tongue effect), that make it an attractive option with which to model episodic memory [2, 3]. SDM is still being implemented for various applications (e.g., [4-6]). Several improvements have been proposed for SDM; for example Ramamurthy and colleagues introduced forgetting as part of an unsupervised learning mechanism [7, 8]. The same authors also proposed the use of ternary vectors, introducing a "don't care" symbol as a third possible value for the dimensions of the vectors [9, 10].

Sequences are important representations for cognitive agents. Agents act over time and cognitive agents adapt *and* act over time. Simple events can be combined into more complex ones forming sequences, or even trees, of simpler events [11-13]. Kanerva, in his original work, described using SDM to store sequences. His procedure has the disadvantage of losing most of the auto-associativeness and noise robustness of the memory. Later he proposed hyperdimentional arithmetic as a new mechanism for storing sequences and other data structures such as sets and records [14]. Even though this new mechanism is an improvement over the original SDM mechanism, it is still

limited in its noise robustness, and is very sensitive to interference (see below). Although interference is a desirable property of the memory because it mimics psychological effects, in this case it diminishes the capacity to retrieve sequences.

Here we propose a variant to the original SDM, called Extended Sparse Distributed Memory (ESDM) that is especially suitable for storing sequences and other data structures such as trees. This new extension considerably improves the performance of sequence storage of the memory as compared to both the original SDM memory sequence storage and the hyperdimensional arithmetic sequence storage version introduced by Kanerva [14].

This paper is a follow up of [15] where we first introduced ESDM. In this present work we expand the description of the algorithm and include the results of several experiments not present in the previous paper. The memory was tested first by storing a large training set and analyzing its noise robustness. Then sequence storing was simulated, and finally an important parameter k (see later) was studied for different scenarios of sequence storage.

In the following section we briefly describe SDM and compare it with other memory models. Then we introduce Extended SDM, discussing several uses of this extension and its results. Several simulations are then presented and discussed. Finally we propose some future directions.

1. Sparse Distributed Memory

Here we present a brief introduction to SDM concepts. Both leisurely descriptions [16] and highly detailed descriptions [1] are available. Readers already familiar with SDM can skip this section.

SDM implements a content addressable random access memory. Its address space is of the order of 2^{1000} or even more. Both addresses and words are binary vectors whose length equals the number of dimensions of the space. This memory is based on the properties of high dimensional spaces. In binary spaces with high dimensionality, on the order of 1,000 or 10,000 dimensions, the distribution of the distances from a point of the space to any other point in the space are highly concentrated at around half of the maximum distance. To calculate distances between two vectors the Hamming distance is used here, but other metrics can be used as well. For example, for a space of 1,000 dimensions, more than 99.9999% of the vectors are at a distance between 422 and 578 from a given vector of the space [1]. A memory with such a high dimensional address space is impossible to build. The number of locations in such an address space can be compared with the number of atoms in the universe [16]. Thus, to construct the memory, a sparse, uniformly distributed sample of addresses is chosen. In our example, the sample size is on the order of 2^{20}. These addresses are called hard locations. The number of hard locations, also the size of the memory, is denoted by s. The distribution of the hard locations need not be uniform. For example Ratitch and Precup [17] created the hard locations as needed, distributing the hard locations following the distribution of the data. This design does not require allocating memory for hard locations that are not used, as is done in the original SDM. Also Fan and Wang [18], and Anwar and colleagues [19] used genetic algorithms to distribute the hard locations.

Only hard locations can store data. Several hard locations participate in the storing and retrieving of any single word of data. Each hard location has a fixed address, and contains one counter for each dimension. In our example, each hard location has 1,000

counters. A counter is just an integer counter that can be incremented or decremented in steps of size one; they have a range of -40 to 40 for our example. Kanerva [1] proved that this range of the counters is enough, considering the capacity of the memory, for the size of the memory in the example. For other sizes this range may vary.

To write a word vector in a hard location, for each dimension, if the bit of this dimension in the word is 1, the corresponding counter is incremented. If it is 0, the counter is decremented. To read a word vector from a hard location, we compute a vector such that, for each dimension, if the corresponding counter in the hard location is positive, 1 is assigned to this dimension in the vector being read, otherwise 0 is assigned.

When a vector is written to an address in the SDM, it is stored in several hard locations. In the same way, to read from an address in the SDM, the output vector is a composition of the readings of several hard locations. An *activated* hard location is one that participates in a reading or writing operation. To determine which hard locations are activated, i.e. used to read or write, an access sphere is defined. The access sphere for an address vector is a sphere with center at this address that on average encloses a proportion p of all the hard locations of the memory. Notice that p is also the probability of activation of a hard location. The radius of the access sphere depends on the number of dimensions of the space and the probability of activation p. For example, for a SDM with 1,000 dimensions and p equal to 0.1%, the radius of the access sphere is 451. In the example, the access sphere will contain any hard location whose address is less than 451 away from the address vector.

The activation of the hard locations can be achieved using other strategies. Jaeckel [20, 21] proposed several alternatives for their activation, and Karlsson [22] introduced a fast activation mechanism based on Jaeckel's work.

To write a word vector in any address of the memory, the word is written to all hard locations inside the access sphere of the address. To read from any address, all hard locations in the access sphere of the address vector are read and a majority rule for each dimension is applied, i.e., the output vector will have a value equal to 1 at a particular dimension if the majority of the vectors read from the hard locations in the access sphere have a 1 in that dimension, or a value equal to 0 otherwise.

Several authors have studied the capacity of SDM: Kanerva [1, 23], Chou [24] and Keeler [25]. In particular Keeler compared the capacity of SDM with the capacity of a binary Hopfield net. He showed that both memories have the same capacity per storage element or counter. However, SDM presents an interesting advantage over Hopfield nets. In the former, the size of the words is independent of the number of storage elements; on the other hand, in the Hopfield nets the size of the words determines the capacity of the memory. Doubling the hard locations in SDM doubles the capacity of the memory for a given word size [23].

Willshaw networks [26], can achieve an information capacity of 0.69, which is higher than for SDM. However, this Willshaw maximum capacity can only be achieved for very sparse vectors, i.e. vectors in which the number of 1's are much less than n, where n is the dimensionality of the vector. Knoblauch and colleagues [27] extensively analyzed the performance of Willshaw networks and pointed out the importance of relating the capacity of associative memories with their fidelity. Comparing SDM with Willshaw networks in these terms can be interesting. However it is outside of the scope of this work.

SDM can be seen as a synchronous, fully connected, three-layer, feed-forward artificial neural network [23]. The input layer is just the input vector. The hidden layer

corresponds to a vector of size s that represents the active hard locations. The matrix composed by the hard locations' addresses corresponds to the matrix of synaptic weights between the input and hidden layers. The output layer is the output vector. Finally, the matrix of synaptic weights between the hidden and output layers is determined by the counters of the hard locations. In practice, due to the mechanism and characteristics of SDM, its training is faster than that of back propagation networks. Even one-shot learning is possible using SDM.

The complexity of the SDM algorithm is dominated by the activation of the hard locations. It has a time complexity $O(sn)$, where s is the size of the memory and n the number of dimensions. As discussed above, the capacity of SDM linearly increases with s. Thus, the time increment due to increasing the capacity of the memory is linear even for large values of s. Nevertheless, the time complexity can be improved using the Karlsson [22] design. Also SDM can be easily implemented as a parallel algorithm or even implemented in hardware [23].

In general, the SDM is used as an auto-associative memory, so the address vector is the same as the word vector. In this case, after writing a word in the memory, the vector can be retrieved using partial or noisy data. If the partial vector is inside a critical distance from the original one, and it is used as address with which to cue the memory, the vector read will be close to the original one. This critical distance depends on the number of vectors already stored in the memory. If the process is repeated, using the first recovered vector as address, the new reading will be even closer to the original. After a few iterations, typically fewer than ten, the readings converge to the original vector. If the partial or noisy vector is farther away than the critical distance, the successive readings from the iterations will diverge. If the partial vector is about at the critical distance, the iterations yields vectors that are typically at the same critical distance from the original vector. This behavior mimics the "tip of the tongue" effect.

When storing sequences of vectors in this SDM, the address cannot be the same as the word, as it is in the auto-associative case. The vector that represents the first element of the sequence is used as address to read the memory. The vector read is the second element in the sequence. This second vector is used as address to read the memory again to retrieve the third element. This procedure is repeated until the whole sequence is retrieved. The problem with this way of storing sequences is that it is not possible to use iterations to retrieve elements of the sequence from noisy input cues. So, the memory is far less robust.

Kanerva [14] introduced hyperdimensional computing, based on large binary vectors, as an appropriate tool for cognitive modeling, including holistic representation of sets, sequences and mappings. Among the various vector operations proposed, multiplication of binary vectors by bitwise xor, permutation, and sum with normalization are relevant to the present work, and will be discussed here.

When two binary vectors are combined using bitwise xor, the result of this operation is a new vector of the same dimensionality as the original ones. This operation has several interesting properties. First, the resulting vector is dissimilar to the two original ones, i.e. farther than the critical distance. Second, the xor operation is reversible. Third, this operation preserves Hamming distances.

Permutation is an operation that shuffles the positions (dimensions) of one vector. Mathematically, this corresponds to multiplying the vector by a square matrix M with one 1 in each row and column while the other positions contain 0. This operation is also reversible, multiplying by M^T, and it preserves Hamming distances as well.

Finally, the sum operation is the arithmetic (integer) sum of the values of each dimension of two or more vectors. For this operation the bipolar representation of the vectors is used, i.e., the value 0 is replaced by -1. The resulting vector is an integer vector. To transform this vector into a binary vector, a normalization operation is required. If one dimension has a positive value, the normalized binary vector has a 1 in this dimension. If the value is negative, the normalized vector has a 0 in this dimension. Ties are resolved at random. The sum with normalization has interesting properties: the resulting vector is similar to each of the vectors summed up; i.e. the distance between them is less than the expected distance between any two vectors in the space. Also, xor multiplication distributes over the sum.

Based in these properties, it is sometimes possible to retrieve the individual added vectors from the sum vector. This is feasible only if the number of vectors added is small, i.e. three or fewer vectors. Even with this small number, the interference among the vectors in the sum makes the retrieval of the original vectors from the sum not very reliable.

Kanerva describes how to store sequences of vectors using hyperdimensional arithmetic [14]. We will briefly describe this procedure and compare it with our implementation in section 3. The main problem with this procedure is that it uses the sum operation, and so shares the same problems with the sum mentioned above while reconstructing the sequence. Also it uses permutation, and as we discussed before, this operation requires matrices that are outside of the binary vector domain.

2. Extended SDM

Here we present a novel structure, built upon SDM, that we call extended sparse distributed memory (ESDM). The main idea of this new memory structure is the use of vectors with different lengths for the addresses and the words. A word has a longer length than the address in which it is stored. Each address has n dimensions while each word has m dimensions with $n<m$. Moreover, the address vector is included in the word vector (See Figure 1). Formally, in a word of length m and with an address with length n, the first n bits of the word compose the address.

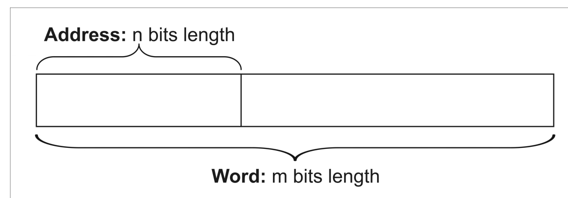

Figure 1 A word vector with its address section.

The structure of this new memory system is similar to the original SDM. It is composed of hard locations, each of which has an address and counters. The address is a fixed vector of length n. But each hard location has m counters, where m is greater than n. To store a word vector in the memory, the procedure is the same as described for SDM, except that now the first n bits of the word are used as address. To read from an address in the memory, again the procedure is similar to the one used for SDM.

During each iteration, a word is read from the memory and its first *n* bits are used to read in the next iteration.

Formally, the address vector is A = $(WM)^T$, where A is an address vector of size *n*, W is the word vector of size *m* and M is a *n* x *m* rectangular diagonal matrix with all 1s in the diagonal.

It is important to notice that the whole word vector, including the address, comprises the useful data. Conceptually, this memory is a mix of auto-associative and hetero-associative memories. The address part of the word is auto-associative whereas the rest of the word is hetero-associative. This allows us to preserve, and even to improve, the desirable characteristics of the SDM. First, with an initial vector as address to cue the memory, it is possible to retrieve the corresponding word, even if the initial vector is a noisy version of the stored one. This means that ESDM maintains the noise robustness characteristic of SDM. Second, the data of each vector is stored in a number of hard locations in a distributed way. So, it is also robust in the case that some hard locations are corrupted or lost. Third, the previously discussed psychological characteristics in SDM are also present in ESDM. Finally, the hetero-associative part of the words in ESDM allows storing other data related with the address data but without interfering with it. This is a notable improvement over the original SDM that relies on the flawed sum operation to achieve the same goal but with far less effectiveness.

Lawrence and Trappenberg found similar conclusions with different associative memory architectures [28]. They studied the advantages of using a combination of auto-associative and hetero-associative neural networks especially for sequence learning. In particular, they emphasized the importance of both the auto-associative and hetero-associative parts to achieve robust sequence memory. The auto-associative part is important for noise robustness allowing cueing the memory with partial or noisy inputs, whereas the hetero-associative part points to the next element in the sequence.

3. Storing sequences and other data structures

In section 1 we mentioned two approaches suggested by Kanerva [1, 14] for storing sequences in SDM. We also mentioned that both approaches have important disadvantages that weaken the auto-associatively, content addressability and noise robustness properties of the memory.

Using associative memories for sequence storage has been long studied. Wang and Yuwono [29] described the problems of using several types of neural networks to store sequences, including Hopfield and Willshaw. Stringer et. al. [30] studied hetero-associative continuous attractor networks to solve path-integration. We have already mentioned the results of Lawrence and Trappenberg [28]; they also provided a good review of associative sequence models. Finally several authors have proposed variations of Hopfield and other memories to store sequences, for example [31].

The implementation of sequence storing in ESDM is straightforward and eliminates the disadvantages mentioned. The most basic implementation uses addresses of length *n* and words of length 2*n*, as shown in figure 2. The sequence is composed of vectors of length *n*. To store the sequence, the first two vectors E_1 and E_2 are concatenated forming a word of length 2*n*. We will say that the word has two *sections* of *n* bits each. This word is stored in address E_1. Then E_2 and E_3 are concatenated and stored in address E_2. The process continues until the full sequence is stored. A special vector can be used to indicate the end of the sequence.

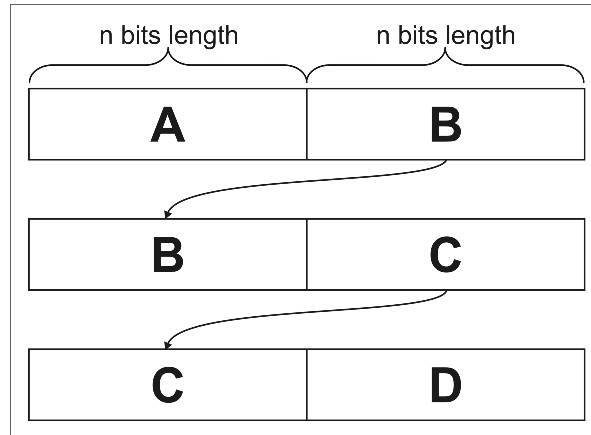

Figure 2 Basic sequence representation using 2n word vectors

To retrieve the sequence, the initial vector of the sequence is used to read a word from the memory. This word is divided in two sections. The second section is the second vector in the sequence. Repeating this procedure, the whole sequence is retrieved. Notice that in each reading during the retrieval of the sequence, the vector used as address can have some noise, but the iterating reading from the memory cleans it up, as explained previously. One problem with this implementation occurs when two sequences are stored in the memory that share a common vector. For example:

AB**CD**E and FG**CH**I

In the example, the word CD is stored in address C but the word CH is stored in C also. This produces the undesirable interference between D and H that prevents the correct retrieval of one or even both of the sequences. One plausible solution is to use the same procedure proposed by Kanerva using hyperdimensional operations. The first reading from the memory again uses the initial vector of the sequence. But the following addresses are calculated using the previously read vectors of the sequence. An elegant combination is achieved using permutation and sum operations [14]. For example if Π denotes a random permutation, then the address for the third element of the sequence is:

(1) $$A_3 = [\Pi(E_1) + E_2]$$

With this address we read the memory, and from the word read the next vector of the sequence, i.e. E_3, is retrieved. The following addresses are calculated in the same way.

(2) $$A_{i+1} = [\Pi(A_i) + E_i]$$

An interesting option is to preserve the sum of the vectors in each reading and multiply it by a scalar *k* between 0 and 1, for example 0.8. This produces an effect of *fading away* of the old vectors of the sequence in the calculation of the next address.

(3) $$A'_{i+1} = k * \Pi(A'_i) + E_i$$

(4) $$A_{i+1} = [A'_{i+1}]$$

Where A' is the real vector with the sum before normalization.

The introduction of the scalar k has another critical function. The normalization required after the sum introduces excessive noise that diminishes the probability of recovering the sequence. See the simulations section below for a discussion of this subject.

These equations can be used in the original SDM, as pointed out by Kanerva. In both situations, operations with sums are used but the advantage of this implementation is that the retrieval of the succeeding vector in the sequence does not depend on operations that extract the vector from the sum. Here the sum is used only to compute the next address, but the vector is extracted directly from the second part of the read word.

In a similar way, other data structures can be stored in ESDM. For example, to store binary trees, addresses of length n and words of length $3n$ are used. With the address of the root of the tree the first word is retrieved. The word is divided into three sections, left, center and right. The left section holds the content of the node in the tree; the center section is used as an address with which to read the left child node of the tree; the right section holds the address of the right child node. This procedure is repeated until the whole tree is retrieved. Notice that here again noisy vectors can be used, and ESDM takes care of cleaning them up. Also, a similar mechanism to the one described for sequences can be used to avoid problems related to repeated vectors in several structures.

Other data structures can be easily derived from sequences and trees. A *double linked sequence* can be constructed adding another section of n bits to the word. The address of the previous element in the sequence is stored there. This allows navigating the sequence in reverse order. Something similar can be used to store the parent of a node in a tree. This allows navigating the tree from the bottom up. Finally, more sections of n bits can be added to each word in the tree so that trees with greater degrees can be stored. Interestingly, a tree can represent a more meaningful data structure, like a record, where each child node represents a field of the record, and the root the record itself. An even simpler representation for record is a word with several sections where each section represents a field of the record.

4. Simulations

For the simulation and testing of the ESDM we implemented the memory using a data base for the main storage of the hard locations, and a RAM cache to speed up the store and retrieve operations. This allows us to create large ESDMs, with millions of hard locations and word dimensions on the order of 1,000 or even 10,000 bits, even using modest computers.

Several simulations were performed with the ESDM. First, the capacity and noise robustness of the extra bits of the words were compared with these same characteristics of the standard SDM. Second, the sequence storage and retrieval were tested for several

values of *k*. Third, retrieving sequences from intermediate elements was analyzed. Finally, recovering sequences that have a common element, i.e. crossing sequences, was tried. In this section we present the details of these simulations, their results and a discussion.

4.1. ESDM capacity and noise robustness

These simulations test the capacity of the memory in comparison to its noise robusteness. Kanerva [1] proved that the critical distance of SDM is a function of the number of words stored in the memory. He also proved that the maximum capacity of the memory is reached when the critical distance reaches zero, and that it is approximately equal to 10% of the number of hard locations for a memory with vectors of 1,000 dimensions. After this number it becomes impossible to retrieve a stored vector even when cueing the memory with the same vector. For a complete analysis of SDM capacity see [23-25]. Reading from ESDM is essentially the same as from SDM, with the discarding of the extra bits of the word. Hence, convergence during a read in ESDM is the same as in SDM, and the critical distance and capacity are also similar to those of SDM. However, we need to show that the percentage of errors (changed bits) in the words read from ESDM is similar to the percentage of errors in the words read from a standard SDM. If only the address part of the vectors stored in ESDM is used, the memory is equivalent to the standard SDM, so the error comparison was performed between the address part and the whole word of the same simulation.

Several simulations were performed to test the percentage of errors in the words read. An ESDM with 200,000 hard locations, an address length of 1,000 dimensions and a word length of 2,000 dimensions (including the address) was used for the simulations. The size of the memory, i.e. number of hard locations, was chosen to have enough hard locations in the access sphere for each read or write to support the desired properties of the ESDM, but to be as small as possible to limit the number of reads and writes required to perceive the effects of loading the memory. The size of the vectors was chosen to match those used by Kanerva [1]. For this particular simulation, a total of 10,000 random vectors were stored in the ESDM, which is roughly half of the memory capacity.

The storing of vectors in the memory was done in stages, writing 1,000 vectors in each stage. At the end of each stage, the vectors were read from the memory. For the readings, 10% of the bits of each vector address were changed randomly, and these noisy vectors were used as cues. Table 1 and Figure 3 show the results of this simulation.

An analysis of the retrieved vectors shows that the proportion of errors for the word and the address is constant and roughly proportional to the difference in size. i.e. a word that is twice as large as the address has twice the number of incorrect dimensions than the address. Also, the percentage of retrieved vectors is consistent with the diminishing of the critical distance as more vectors are stored in the memory [1].

Stage	Retrieved	Iterations		Error mean	
		mean	Std	address	word
1	100.00%	2.59	0.49	0.00	0.00
2	100.00%	3.04	0.24	0.00	0.00
3	99.80%	3.51	0.59	0.00	0.00
4	98.40%	4.31	0.90	0.00	0.00
5	90.30%	5.23	1.25	0.04	0.09
6	71.20%	6.16	1.41	0.20	0.39
7	47.60%	7.30	1.62	1.37	2.83
8	22.30%	8.24	1.58	3.78	6.18
9	15.00%	9.50	1.83	1.15	1.60
10	12.60%	11.09	3.34	1.54	2.47

Table 1. Simulation 1. ESDM capacity and noise robustness. In each stage 1000 vectors were stored. Then the same vectors were retrieved after adding 10% noise to the cue (address). The number of iterations and errors correspond to the retrieved vectors. The address part is equivalent to the standard SDM result.

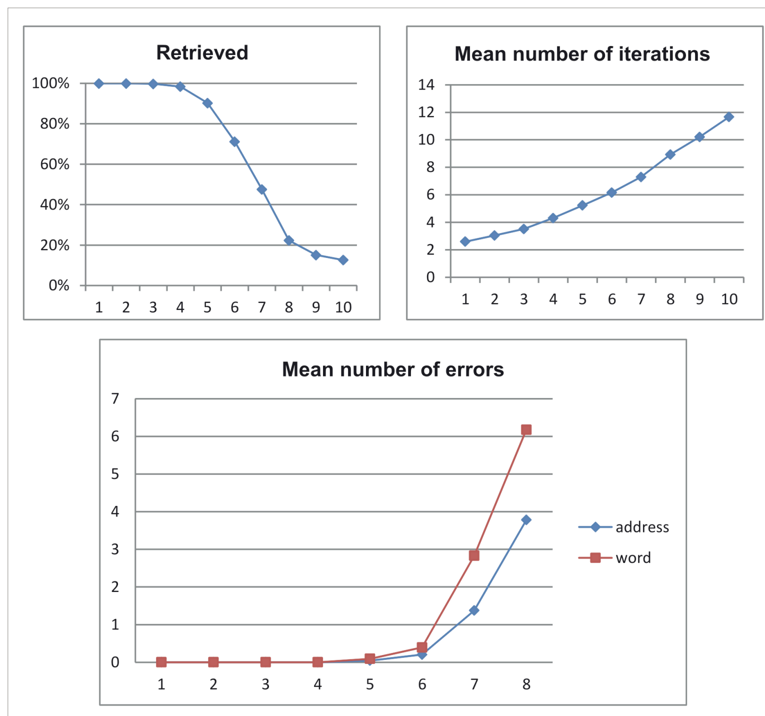

Figure 3 Percentage of retrieved vectors in each stage, iterations (mean) required in each stage, and errors (number of changed bits) of the address part and the whole word of the retrieved vectors in each stage.

Another simulation was performed to show the noise robustness of ESDM. The same ESDM was used as for the previous simulation, with 10,000 vectors already stored in the memory. The vectors were also preserved in a separate database so they could be used as cues or compared with the retrievals from the ESDM. The simulation was performed in three stages. In each stage, one thousand vectors were randomly selected from the set of stored vectors, and the memory was read using the address part of these vectors with an amount of noise. The amount of noise changed in each stage: 0% in the first stage, 5% in the second and 10% in the third. Table 2 summarizes the results of this simulation.

Stage	Noise	Retrieved	Error mean
1	0%	100.00%	0.286
2	5%	97.00%	4.784
3	10%	14.80%	2.439

Table 2. Simulation 2. ESDM capacity and noise robustness. In each stage 1000 vectors were retrieved from a ESDM with 10,000 stored vectors , and an amount of noise was added in the cue (address). The number of errors corresponds to the successfully retrieved vectors and represents the average number of bits changed in each vector.

The number of successful retrievals was high with a small amount of noise, and the error (number of changed bits in the retrieval) was very small, less than a bit on average. Even more, 93.3% of the vectors had 0 errors in stage 1 and 79% of the retrievals in stage 2 had fewer than 5 errors. As expected, the number of retrieved vectors decays abruptly when the vectors used as cues reach the critical distance. The critical distance is the distance from whence the probability to converging to the stored value is 50%. The critical distance is a function of the number of hard locations and the number of stored vectors in the memory. For the ESDM used in this experiment, with a load of 50% of its capacity, distances of 100 bits (10% of the address) from the original vectors are beyond the critical distance (see [1] for details).

4.2. Sequences

We performed several simulations to test sequences stored in ESDM. In each simulation 50 or 100 sequences of 20 elements each were stored. As in the previous simulations, ESDM memories with 200,000 hard locations, an address length of 1,000 dimensions and a word length of 2,000 dimensions (including the address) were used for these simulations. For each simulation a new ESDM was used, the load of the memory in each simulation was between 5% to 10% of the memory capacity. This prevented interference among stored vectors. For these tries, we considered a sequence successfully retrieved if all of their elements were retrieved with a small amount of noise (less than 5%).

The first simulation stored 50 sequences using equation (2). Using the same equation for retrieving, 49 sequences were restored. The same simulation was repeated

with 100 sequences and none of the sequences could be restored. The load of the memory was 10%, so interference does not explain this result. The problem here is the normalization after the sum in equation (2). When the address is calculated the sum has only two binary vectors as operands. When in one dimension the two operands have different values, the value for this dimension is undetermined and then a random value is chosen during the normalization. Assuming that the vectors are uniformly randomly distributed, the average number of bits that are undetermined is 50% and this introduces excessive noise in the address preventing the retrieving of the element.

To avoid this problem, equations (3) and (4) were used. Since one of the operands has a smaller weight than the other, the sum has no undetermined dimensions, and the problem disappears. In a simulation where 100 sequences were stored using equations (3) and (4) with $k = 0.8$ all the sequences were restored with 0 errors in its elements.

The use of the parameter k has other interesting consequences due to the fact that the weight of the previous elements diminishes as the sequence advances. It is possible to “step into” the sequence in the middle. However, more than one element may be required for the cue. For smaller values of k, fewer elements are required as part of the cue to step into the sequence. Conversely, if two (or more) sequences have common elements, the higher the value of k (nearly equal to one) the higher the chance of retrieving the correct sequence. The value of k is then a tradeoff between these two desirable properties.

Several simulations with different values of k were performed. First, the step into property was tested. Three simulations with values of k equal to .7, .8 and .9 respectively were performed. 100 sequences with 20 elements each were stored in each simulation. Then, 10 of the stored sequences were chosen, and for the elements of these sequences, the number of required elements in the cue to be able to step into the sequence at that element was evaluated. To avoid transitory effects, only elements after the fifth element in the sequences were used as points to step into. Table 3 shows the results of these simulations.

Stage	k	Required Elements	
		Average	Std
1	0.7	1.085	0.280
2	0.8	2.697	0.679
3	0.9	6.000	1.265

Table 3. Effect of the parameter k in stepping into the sequence. In each stage, the number of required elements in the cue to step into the sequence at different points was evaluated.

Another series of simulations was performed to evaluate the retrieval of sequences with common elements, i.e. sequence intersection. Four simulations with values of k between .9 and .6 respectively were performed. Ten pairs of sequences with 20 elements each were stored in each simulation. The sequences in each pair had a common element. In every case, the intersection was after the forth element in the

sequences. A number of random vectors were stored in the memory so as to achieve a load of 10% of the capacity of the memory.

Then, each of these sequences was retrieved from the memory and the number of successfully recovered sequences noted. With all of these values of k, all sequences were successfully retrieved. This result shows that the feature of correctly retrieving intersecting sequences is invariant over the value of k. However, equations (3) and (4) suggest that if two sequences have more than one consecutive common element, higher values of k will perform better.

Notice that when k is equal to or less than .5, the first term in equation (3) is always less than 1 and it does not contribute to the final value after normalization in equation (4). As a consequence, the next address is only a function of the previous element, so that most elements after the intersecting element are not able to be retrieved. This is because of the interference produced by the common element of the sequences.

Comparing the results of the last two groups of simulations, a balance between the two characteristics, step into and crossing of sequences is achieved with a value of k between .6 and .8. Of course, the selection of the value of k depends on the requirements of the application of the ESDM.

5. Conclusions

Here we have presented an extension of the original SDM that addresses several of its difficulties with storing compound data structures like sequences, trees and records. Our ESDM preserves the desirable, biologically inspired, properties of the original. It is also still noise robust, auto-associative and distributed. These, combined with the possibility of storing sequences and other compound data structures, make ESDM an even more attractive option with which to model episodic memories.

The simulations carried out successfully tested the performance of the ESDM in several scenarios. The importance of the parameter k was shown not only for the simple storage of sequences but also to achieve desired features of being able to step into in the middle of sequences, and to support common elements in different sequences.

ESDM is compatible with other improvements already studied, such as the introduction of the “don’t care” symbol [9, 10], or the forgetting mechanism [7, 8]. Including this forgetting mechanism is a natural future step for this architecture.

ESDM has the potential for further extensions. Representation of other data structures, and combining them with hyperdimensional vector arithmetic are possible paths for further development.

References

1. Kanerva, P., *Sparse Distributed Memory*1988, Cambridge MA: The MIT Press.
2. Baddeley, A.D., Conway, M., and Aggleton, J.P., *Episodic Memory*2001, Oxford: Oxford University Press. 294.
3. Franklin, S., et al., *The Role of Consciousness in Memory.* Brains, Minds and Media, 2005. **1**: p. 1–38.

4. Furber, S.B., et al., *A Sparse Distributed Memory based upon N-of-M Codes.* Neural Networks, 2004. **17**(10): p. 1437 - 1451.
5. Bose, J., Furber, S.B., and Shapiro, J.L., *Spiking neural sparse distributed memory implementation for learning and predicting temporal sequences.* Lecture Notes in Computer Science, 2005. **3696/2005**: p. 115 - 120.
6. Meng, H., et al. *A modified sparse distributed memory model for extracting clean patterns from noisy inputs*. in *International Joint Conference on Neural Networks (IJCNN)*. 2009. Atlanta, GA, USA.
7. Ramamurthy, U., D'Mello, S.K., and Franklin, S., *Realizing Forgetting in a Modified Sparse Distributed Memory System*, in *Proceedings of the 28th Annual Conference of the Cognitive Science Society*, C. Schunn and S. Lane, Editors. 2006, Lawrence Erlbaum Associates: Mahwah, NJ. p. 1992–1997.
8. Ramamurthy, U. and Franklin, S., *Memory Systems for Cognitive Agents*, in *Proceedings of Human Memory for Artificial Agents Symposium at the Artificial Intelligence and Simulation of Behavior Convention (AISB'11)*2011: University of York, UK. p. 35-40.
9. D'Mello, S.K., Ramamurthy, U., and Franklin, S., *Encoding and Retrieval Efficiency of Episodic Data in a Modified Sparse Distributed Memory System*, in *Proceedings of the 27th Annual Meeting of the Cognitive Science Society. Stresa, Italy*2005.
10. Ramamurthy, U., D'Mello, S.K., and Franklin, S., *Modified Sparse Distributed Memory as Transient Episodic Memory for Cognitive Software Agents*, in *Proceedings of the International Conference on Systems, Man and Cybernetics*2004, IEEE: Piscataway, NJ.
11. Snaider, J., McCall, R., and Franklin, S., *Time Production and Representation in a Conceptual and Computational Cognitive Model.* Cognitive Systems Research, 2012. **13**(1): p. 59-71.
12. Sun, R. and Giles, C.L., *Sequence learning: From recognition and prediction to sequential decision making.* IEEE Intelligent Systems, 2001. **16**(4): p. 67-70.
13. Kurby, C.A. and Zacks, J.M., *Segmentation in the perception and memory of events.* Trends in Cognitive Science, 2008. **12**(2): p. 72-9.
14. Kanerva, P., *Hyperdimensional Computing: An Introduction to computing in distributed representation with high-dimensional random vectors.* Cognitive Computation, 2009. **1**(2): p. 139-159.
15. Snaider, J. and Franklin, S. *Extended Sparse Distributed Memory*. in *Biological Inspired Cognitive Architectures 2011*. 2011. Washington D.C. USA.
16. Franklin, S., *Artificial Minds*1995, Cambridge MA: MIT Press.
17. Ratitch, B. and Precup, D., *Sparse distributed memories for on-line value-based reinforcement learning.* Lecture Notes in Computer Science (LNCS), 2004. **3201**: p. 347-358.
18. Fan, K.C. and Wang, Y.K., *A genetic sparse distributed memory approach to the application of handwritten character recognition.* Pattern Recognition Letters, 1997. **30**(12): p. 2015-2022.
19. Anwar, A., Dasgupta, D., and Franklin, S. *Using Genetic Algorithms for Sparse Distributed Memory Initialization*. in *International Conference Genetic and Evolutionary Computation(GECCO)*. 1999.

20. Jaeckel, L.A., *An Alternative Design for a Sparse Distributed Memory.*, 1989, Research Institute for Advanced Computer Science, NASA Ames Research Center.
21. Jaeckel, L.A., *A Class of Designs for a Sparse Distributed Memory*, 1989, Research Institute for Advanced Computer Science, NASA Ames Research Center.
22. Karlsson, R., *A fast activation mechanism for the Kanerva SDM memory.* Proceedings of the RWC Symposium, 1995: p. 69-70.
23. Kanerva, P., *Sparse Distributed Memory and related models*, in *Associative Neural Memories: Theory and Implementation*, M.H. Hassoun, Editor 1993, Oxford University Press: New York. p. 50-76.
24. Chou, P.A., *The capacity of the Kanerva associative memory.* IEEE Trans. Information Theory, 1989. **35**(2): p. 281-298.
25. Keeler, J.D., *Comparison between Kanerva's SDM and Hopfield-type neural networks.* Cognitive Science, 1988. **12**: p. 299-329.
26. Willshaw, D.J., *Holography, associative memory, and inductive generalization*, in *Parallel Models of Associative Memory*, G.E. Hinton and J.A. Anderson, Editors. 1981, Erlbaum: Hillsdale, N.J. p. 83-104.
27. Knoblauch, A., Palm, G., and Sommer, F.T., *Memory Capacities for Synaptic and Structural Plasticity.* Neural Computation, 2010. **22**(2): p. 289-341.
28. Lawrence, M., Trappenberg, T., and Fine, A., *Rapid learning and robust recall of long sequences in modular associator networks.* Neurocomputing, 2006. **69**(7-9): p. 634-641.
29. Wang, D. and Yuwono, B., *Incremental Learning of Complex Temporal Patterns.* IEEE Transactions on Neural Networks, 1996. **7**(6): p. 1465--1481.
30. Stringer, S.M., et al., *Selforganizing continuous attractor networks and motor function.* Neural Networks, 2003. **16** (2): p. 161-182.
31. Maurer, A., Hersch, M., and Billard, A.G., *Extended hopfield network for sequence learning: Application to gesture recognition.* Proceedings of the ICANN 2005, 2005.