Integer Sparse Distributed Memory

Javier SNAIDER, Stan FRANKLIN

Computer Science Department & Institute for Intelligent Systems, The University of Memphis
FedEx Institute of Technology, 365 Innovation Dr., Memphis, TN 38152
jsnaider@memphis.edu, franklin@memphis.edu

Abstract

Sparse distributed memory is an auto-associative memory system that stores high dimensional Boolean vectors. Here we present an extension of the original SDM, the Integer SDM that uses modular arithmetic integer vectors rather than binary vectors. This extension preserves many of the desirable properties of the original SDM: auto-associativity, content addressability, distributed storage, and robustness over noisy inputs. In addition, it improves the representation capabilities of the memory and is more robust over normalization. It can also be extended to support forgetting and reliable sequence storage.

Introduction

Sparse distributed memory (SDM) (Kanerva, 1988) is based on large binary vectors, and has several desirable properties. It is distributed, auto-associative, content addressable, and noise robust. Moreover, this memory system 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 (Baddeley, Conway & Aggleton, 2001; Franklin et al, 2005). Implementations of SDM are ongoing for various applications (e.g., Furber et al, 2004; Meng et al, 2009; Mendes, Coimbra & Crisostomo, 2009; Jockel, 2009). Several improvements and variations have been proposed for SDM; for example Ramamurthy and colleagues introduced forgetting as part of an unsupervised learning mechanism (Ramamurthy, D'Mello & Franklin, 2006). 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 (D'Mello, Ramamurthy & Franklin, 2005). Also Jaeckel (1989a, 1989b) proposed two variations of the original SDM, the selected coordinate design and the hyperplane design. Both designs modify the way that hard locations (see next section) are selected. These designs slightly improve the signal to noise ratio of the memory. Furber and colleagues (2004) created a combined version of the Jaeckel's hyperplane design and a correlation matrix memory using sparkling neurons.

The original SDM uses binary vectors for both addresses and data, i.e. words. This usage results in several limitations. First, real data is not always Boolean, making representations using more than two values desirable. A possible solution is to use several dimensions of the word vectors to represent one feature, but this approach does not fit very well with the structure of SDM. In the distance calculation, difference in any dimension weights the same as any other dimension, but if several bits, i.e. dimensions, are used to represent a single feature, the weight of the bits should not be the same.

Mendes and colleagues (Mendes, Coimbra & Crisostomo, 2009) evaluated several binary encodings to use with SDM in robot navigation tasks, and reported their difficulties and limitations. Using Natural BC coding some transitions have Hamming distances that incorrectly reflect the difference between the features. For example, the Hamming distance between seven (0111) and eight (1000) is 4 instead of 1, which is desired. They also reported the performance of the Gray code, which only partially mitigates this effect. The best solution that they proposed is to use a sum code, that is a base one code where, for example, 3 is represented as 111 and 5 as 11111. This coding substantially increases the dimensionality of the memory. Interestingly, they report that grouping bits and processing them as integers produces excellent performance. However, their implementation diminishes some of the desirable properties of SDM. The extension proposed in this paper directly uses integers vectors, achieving similar performance but without the disadvantages reported by Mendes.

Another disadvantage of binary vectors is the loss of information due to the noise introduced into the representation by the normalization used in combining vectors. Vectors can be summed up dimension by dimension (for this operation, vectors belonging to $\{-1; +1\}^n$ are used). This operation produces a vector belonging to $\mathbb{Z}^n$. The normalization process reduces the resultant to a vector that is also in $\{-1; +1\}^n$ but with significant loss of information. See for example (Kanerva, 2009; Snaider & Franklin, 2011).

Here we propose a new version of SDM, the Integer Sparse Distributed Memory (Integer SDM). This version is based on large vectors where each dimension has a range of possible integer values. The memory has properties similar to the original one, noise robustness, auto-associativity and being distributed. In addition, this memory avoids the limitations imposed by binary representation, as described above.

Sparse Distributed Memory

Being based in the structure and behavior of the original SDM, it is better to describe Integer SDM using concepts from that original. In this section, we first briefly describe the components of SDM that are similar to those used in Integer SDM. For more information about SDM, both leisurely descriptions (Franklin, 1995, pp. 329-344) and highly detailed descriptions (Kanerva 1988) are available.

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. An important property of such high dimensional spaces is that two randomly chosen vectors are relatively far away from each other, meaning that they are uncorrelated. In our example, we will think of bit vectors of 1,000 dimensions. To calculate distances between two vectors in this space, the Hamming distance is used. To construct the memory, a sparse uniformly distributed sample of addresses, on the order of 2^{20} of them, is chosen. The number of addresses selected to construct the memory is denoted by m. These addresses are called hard locations. Hard locations are the units of storage of the memory. Only hard locations can store data. For this purpose, each hard location has counters, one for each dimension. 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.

A hard location can store several words but as a combination of them. In order to be able to reconstruct the original word, many hard locations participate in the storing and retrieving of any single word of data. To read from an address in SDM, the output vector is a composite of the readings of several hard locations. To determine which hard locations are 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, enclosing, on average, a proportion p of the memory's hard locations; in our example 0.1% is used. 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.

In general, the SDM is used as an auto-associative memory, so the address vector is the same as the word vector (but see Snaider & Franklin, 2011). 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 output vector 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 less than ten, the readings converge to the original vector. If the partial or noisy vector is farther than the critical distance away from the original one, the successive readings from the iterations will rapidly diverge.

Integer Sparse Distributed Memory

The structure of Integer SDM is similar to that of SDM. The words and addresses used by Integer SDM are large vectors of integers, i.e. vectors with a large number of dimensions. The possible values for each dimension are in a defined integer range. For example, the range of values can be [-8, 7] or [0, 15]. Any range of values is possible. For simplicity, we will work with ranges with 0 as lower bound and r - 1 as upper bound. There is no limit for the size of the range. However, the storage requirement increases proportionally with the size of the range. More formally, Integer SDM works within multidimensional space with vectors $v \in \mathbb{Z}_r^n$, where n is the number of dimensions of the space and r is the size of the range of values for each dimension. The dimensions of the space follow modular arithmetic, i.e. the values wrap around after r. The greatest possible value for a dimension is r - 1 and the next value after r - 1 is 0.

Integer SDM is composed of hard locations. As in SDM, a small, uniformly distributed, fraction of all possible addresses $a \in \mathbb{Z}_r^n$ are chosen for the addresses of the hard locations. Each hard location has a fixed address and counters, resembling the structure of SDM. However, Integer SDM has a different arrangement of counters: each dimension has r counters, one for each possible value in that dimension. We define c_i as the group of counters corresponding to the dimension i, and $c_i^{(v)}$ as the counter corresponding to dimension i and value $v \in \{0, \dots, r-1\}$. The procedures to read from or write to the memory are similar to the ones used for SDM.

To read or write a word w, first the access sphere of the address is determined. The distance used here is an extension of the Euclidean metric. The distance between two vectors is defined as:

$$d(u,v) = \sqrt{\sum_i (\Delta_i)^2}$$

where: $\Delta_i = \min\left(mod_r(u_i - v_i), mod_r(v_i - u_i)\right)$

Since each dimension in the space follows modular arithmetic, each dimension is like a circle and there are two possible *paths* in dimension i between the values u_i and v_i. Notice that Δ_i is the smaller length of these two paths.

The radius of the access sphere is defined in such a way that on average it encloses a small proportion p of the total number of hard locations m. The access sphere encloses pm hard locations. This value p is also the probability of activation of one hard location, i.e. the probability of one hard location participates in one particular reading or writing operation. For writing the word w in the memory, the counters of every dimension of each hard location in the access sphere are updated using the following rule:

$$c_i^{(v)} \text{ is incremented } \Leftrightarrow v = w_i$$

where w_i is the value of the dimension i of the word w. Notice that only one counter out of r of each dimension of each hard location in the access sphere is incremented.

To read from the memory, first the hard locations in the access sphere are determined. Then the counters of each value of each dimension of all hard locations in the access sphere are summed up:

$$s_i^{(v)} = \sum_{HL \in A.S.} c_i^{(v)}$$

where $s_i^{(v)}$ is the sum of the counters for dimension i and value v. Finally, for each dimension a majority rule is applied among the values:

$$z_i = idx(v) \; of \; \max\left(s_i^{(0)} .. s_i^{(r-1)}\right)$$

where z_i is the value of dimension i of the output vector. This vector z can be used as an address to read again from the memory, iterating in the same way that was described for the original SDM.

The fidelity of the memory, i.e. the probability of retrieving a written word, is better than the original SDM. This improvement in the fidelity is due to the more precise storage in each hard location. Suppose the stored value for dimension i of word w is k, that is $w_i = k$. To incorrectly read w_i from memory, at least one of the sums $s_i^{(v)}$ for the incorrect values ($v \neq k$), must be greater than $s_i^{(k)}$. The value of the sums for incorrect values is due to the contribution of other words written in the memory that share some of the same hard locations used to store w. Assuming the other words written in the memory are uniformly distributed in the space[1], the noise produced by the interference of these written words is distributed in r possible values. This diminishes the expected value and variance of the $s_i^{(v)}$ *for* $v \neq k$. Then the probability of having at least one $s_i^{(v)} > s_i^{(k)}$ is less than in the original SDM for the same number of words stored in the memory. This increment in the fidelity of the memory also increments its capacity: more words can be stored before the effect of interference is noticed. This compensates for the additional requirements of memory storage of this memory compared to SDM.

The complexity of the reading (or writing) operation of the memory is $O(mn + prmn)$. The first term corresponds to the calculation of the distance from w to each hard location, and the second term corresponds to the reading (or writing) of the counters in the hard locations. Since $pr << 1$, the first term dominates. Since the number of hard locations m can be large, the implementation could be slow. However, the algorithm is easily parallelizable to be executed in multithreading or SIMD (e.g. using GPUs) architectures. Moreover, other methods to activate the hard locations, instead of the access sphere, were studied for SDM, and can be used with Integer SDM also. See for example (Jaeckel, 1989a, 1989b). These alternatives would greatly reduce the time complexity of the algorithm.

Experiments and Results

Several simulations were performed to test the percentage of errors in the output words. We used an Integer SDM with 100,000 hard locations and a word length of 1,000 dimensions, where $r = 16$ (i.e. range: [0 – 15].) We used a probability of activation $p = 0.001$, that corresponds to a radius of the access sphere of 188. 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 Integer SDM, but to be as small as possible so as to limit the number of reads and writes required to perceive the effects of loading the memory. A total of 5,000 random vectors were stored in the Integer SDM. The vectors were also preserved in a separate database so they could be used as cues or compared with the retrievals from the Integer SDM.

The simulation was performed in four stages. In each stage, one hundred vectors were randomly selected from the set of 5,000 stored vectors, and the memory was cued using these vectors with some amount of noise, that is with some number of randomly selected dimensions that were changed from the original. The amount of noise changed in each stage was: 5, 10, 20, and 30 percent respectively. In stages 1 and 2, 100% of the vectors were retrieved. Stage 3 had only one retrieval error, and stage 4 produced 65% correct retrievals. The same experiment using the Manhattan distance had similar results: 100% of the vectors correctly retrieved in stages 1, 2, and 3, and 65% in stage 4. The graceful degradation in the performance shown in these experiments is similar to the one observed in the original SDM (Kanerva, 1988).

Another experiment demonstrated the generalization characteristics of the memory. Figure 1a. depicts twelve gray scale (16 levels) images of 33 x 33 pixels each. For each image, one vector of 1,089 dimensions representing the information of the image was stored in the memory.

[1]This assumption is reasonable to give an estimation of the capacity of the memory. However, the memory can store vectors even if they are not uniform distributed, but the capacity will be diminished. See (Kanerva, 1988) for a similar analysis for SDM.

Each of these vectors was saved in the memory only once. The memory used for this experiment is similar to that used in the previous experiment. It has 100,000 hard locations with addresses of 1,089 dimensions, r = 16 and p = 0.001. The memory was then cued using the new vector corresponding to figure 1b. The image of the output vector displayed in figure 1c, which is not in the training set either, is the result of the interference of the stored vectors. Based on this and other characteristics of the memory, Integer SDM is a good candidate to model various memories in cognitive architectures (Ramamurthy & Franklin, 2011).

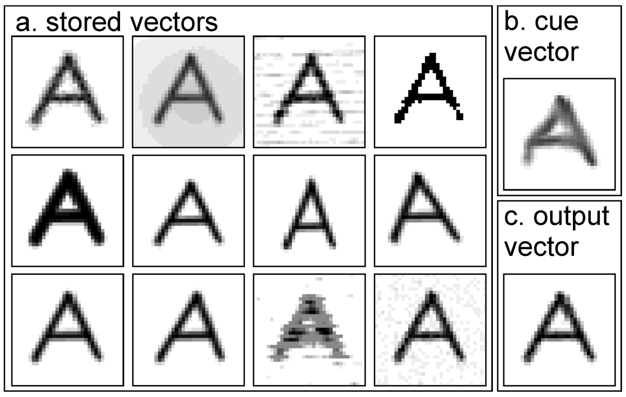

Figure 1. Generalization and pattern formation. a: Images corresponding to the training set vectors. b: Image of the vector used as a cue. c: Image corresponding to the output vector using (b) as cue. Vectors of images (b) and (c) are not in the training set (a).

Conclusions

Here we have presented a new version of SDM, the Integer SDM, that overcomes the limitations of the original SDM resulting from its use of binary vectors. This memory preserves the desirable, biologically inspired, properties of the original. It is also noise robust, auto-associative and distributed. It degrades gracefully when the memory approaches its maximum capacity. It is also able to generalize patterns due to interference of several similar vectors. These properties make Integer SDM a good candidate for modeling episodic memory in autonomous agents.

The integer representation has several advantages over the binary one. The encoding of values is simpler, avoiding undesirable effects of other encodings (Mendes, Coimbra & Crisostomo, 2009; Jockel, 2009), and it diminishes the effect of normalization when several vectors are combined, for example in the storing and retrieval of sequences (Snaider & Franklin, 2011).

Integer SDM is compatible with other improvements already studied for SDM, such as the forgetting mechanism (Ramamurthy, D'Mello & Franklin 2006). Other designs of activation of hard locations, like Jaeckel's selected coordinate design (1989a), can also be implemented with this memory. Another extension, which we have already implemented, applies the same concepts as in Extended SDM (Snaider & Franklin, 2011; Snaider & Franklin, in press) that dramatically improve the capability for storing sequences.

References

Baddeley, A., Conway, M., & Aggleton, J. 2001. *Episodic Memory*. Oxford: Oxford University Press.

D'Mello, Sidney K., Ramamurthy, U., & Franklin, S. 2005. 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*.

Franklin, S. 1995. *Artificial Minds*. Cambridge MA: MIT Press.

Franklin, S., Baars, B. J., Ramamurthy, U., & Ventura, M. 2005. The Role of Consciousness in Memory. *Brains, Minds and Media,* 1: 1-38.

Furber, S. B., Bainbridge, W. J., J.M., C., & Temple, S. 2004. A Sparse Distributed Memory based upon N-of-M Codes. *Neural Networks,* 17(10): 1437-1451.

Jaeckel, L. A. 1989a. *An Alternative Design for a Sparse Distributed Memory.* (No. 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.* (No. Report RIACS TR 89.30): Research Institute for Advanced Computer Science, NASA Ames Research Center.

Jockel, S. 2009. *Crossmodal Learning and Prediction of Autobiographical Episodic Experiences using a Sparse Distributed Memory*. Doctoral Thesis, University of Hamburg, Hamburg.

Kanerva, P. 1988. *Sparse Distributed Memory.* Cambridge MA: The MIT Press.

Kanerva, P. 2009. Hyperdimensional Computing: An Introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation,* 1(2): 139-159.

Mendes, M., Coimbra, A., & Crisostomo, M. 2009. Assessing a Sparse Distributed Memory Using Different Encoding Methods. *Proceedings of the World Congress on Engineering,* 1: 1-6.

Meng, H., Appiah, K., Hunter, A., Yue, S., Hobden, M., Priestley, N., et al. 2009. *A modified sparse distributed memory model for extracting clean patterns from noisy inputs.* Paper presented at the International Joint Conference on Neural Networks (IJCNN), Atlanta, GA, USA.

Ramamurthy, U., D'Mello, S. K., & Franklin, S. 2006. Realizing Forgetting in a Modified Sparse Distributed Memory System. In C. Schunn & S. Lane (Eds.), *Proceedings of the 28th Annual Conference of the Cognitive Science Society,* 1992-1997. Mahwah, NJ: Lawrence Erlbaum Associates.

Ramamurthy, U., & Franklin, S. 2011. *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).

Snaider, J., & Franklin, S. 2011. *Extended Sparse Distributed Memory.* Paper presented at the Biological Inspired Cognitive Architectures 2011, Washington D.C. USA.

Snaider, J., & Franklin, S. in press. Extended Sparse Distributed Memory and Sequence Storage. *Cognitive Computation.*