Sparse Distributed Memory for Binary Sparse Distributed Representations

Ruslan Vdovychenko*
V.M. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine

Vadim Tulchinsky
V.M. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine

ABSTRACT

Sparse Distributed Memory (SDM) and Binary Distributed Representations (BDR), as two phenomenological approaches to biological memory modelling, have a lot of common features. The idea of their integration in a hybrid semantic storage model with SDM used as the low (brain cell) level cleaning memory for BDR used as the high-level symbolic information coder seems natural. The hybrid semantic storage must be able to memorize holistic data (like the structures of interconnected and serialized key-value pairs) in a neural network. It has been proposed several times since 1990th. However, the earlier proposed models are not practical because of insufficient scalability and/or low storage density. The gap between SDM and BDR can be filled using the results of a 3rd theory dealing with sparse signals: Compressive Sensing or Sampling (CS). Such a hybrid semantic storage model is presented. We call it CS-SDM to reflect using a new CS-based SDM design as the cleaning memory for a Binary Sparse Distributed Representation (BSDR) of the holistic data. CS-SDM has been implemented on GPU and demonstrated much better capacity and denoising capabilities than classical SDM designs.

CCS CONCEPTS

• **Mathematics of computing**; • **Coding theory**; • **Information systems**; • **Probabilistic retrieval models**;

KEYWORDS

Sparse Distributed Memory, Compressive Sensing, Compressive Sampling, Binary Sparse Distributed Representations, Vector Symbolic Architecture, associative memory, neural networks, GPU

ACM Reference Format:
Ruslan Vdovychenko* and Vadim Tulchinsky. 2022. Sparse Distributed Memory for Binary Sparse Distributed Representations. In *2022 7th International Conference on Machine Learning Technologies (ICMLT) (ICMLT 2022), March 11–13, 2022, Rome, Italy.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3529399.3529441

1 INTRODUCTION

There are many types of memory known in nature for animals and humans: motor, sensory, semantic short-term and long-term, immune, etc. None of them works similarly to the memory of electronic devices: new information does not overwrite old information, damage to one or a small number of cells (e.g. brain neurons) does not affect the functionality of the entire system, there is no clear separation of "addresses" and "data". The phenomenological approach to studying natural objects is based on building functional models able to simulate the behaviour and features of the objects ignoring their real construction. Among the phenomenological models of natural memory, Sparse Distributed Memory (SDM) model proposed by Pentty Kanerva [1-3] in 1986 is of special interest. Despite the fact that Kanerva used non-biological tools (binary vectors, counters, addresses), later research has shown the SDM structure proximity to the biological memory mechanisms. It possesses many features of natural memory: associativity, generalization ability, sequence memorization, and even making mistakes. A few years later, Lewis Jaeckel has published the hyperplane design of SDM [4, 5]. Its technical advantages are less memory consumption and faster reading/writing operations because of reducing the number of comparisons. Jaeckel's SDM was found more compliant with the mammalian cerebellum functioning (according to the model independently developed by David Marr in 1969 and James Albus in 1975 [6]). Yet the original Kanerva's design is more similar to the immune memory operating (according to the model of Derek Smith, Stephanie Forrest, and Alan Perelson, 1996 [7]).

Another direction of phenomenological research of memory is focused on modelling the holistic information encoding in simple neural structures. This research direction is usually associated with the concept of Vector Symbolic Architecture (VSA) formed at the turn of the 2000s [8]. The general requirements of VSA are very similar: local error tolerance, the ability to express complex relationships such as hierarchical, key-value type, symbolic sequences, etc., by simple means. The similarity of the requirements results in the method similarity: both directions use distributed representations, sparsity and long vectors. Essential VSA results for real vectors were obtained by Fodor and Pylyshyn [9], Smolensky [10], Plate [11]. For binary vectors, the results of Kanerva [12, 13], Sjödin [14], Gayler [15], Rachkovskij [16, 17] and their colleagues had the most significant impact. The term VSA itself was proposed in 2003 by Gayler [18]. Sparse representations attract interest due to their compliance with the nature of the activity of neurons [19]. Hereafter sparse vectors are those where the number of non-zero components is significantly less than the number of zeros. The binary vectors of approximately equal probabilities of 0 and 1 component values are called dense. Methods for transforming real vectors and hierarchical into binary sparse codevectors are well developed today for Binary Sparse Distributed Representations (BSDR) [20].

Binary Distributed Representations of all types use so-called cleaning memory to restore the original codevectors from their

ICMLT 2022, March 11–13, 2022, Rome, Italy

ACM ISBN 978-1-4503-9574-8/22/03…$15.00
https://doi.org/10.1145/3529399.3529441

noisy instances obtained from the VSA component extraction operations (unbinding, probing). Gunnar Sjödin has shown [21] that SDM is not scalable if used as the cleaning memory for dense vectors since it has a high sensitivity to errors in the address. With a constant probability of a 0/1 bit of the address and constant load (the number of recorded vectors per physical memory cell), the signal-to-noise ratio tends to zero with the growth of the number of physical cells. This is another reason for using BSDR in the context of SDM cleaning memory. However, SDM is ill-suited for storing sparse vectors. In this case, most address space is not used, physical memory is used inefficiently, and the reading operation becomes unreliable and complicated even for the known priory probabilities.

As a result, VSA uses conventional computer memory instead of SDM neural memory, and there is a lack of symbolic data representation mechanisms to develop SDM for memorizing semantics. The main contribution of our CS-SDM is overcoming the listed shortcomings and filling the gap by applying the Compressive Sensing (CS) results.

Compressive Sensing (or Compressive Sampling) is a theory started in 2004-2006 by Emmanuel Candes and Terence Tao [22, 23]. Its idea is the ability to solve undetermined systems of linear algebraic equations with normalized coefficients if looking for either a solution of the known number of non-zero components s or the most sparse solution. In other words, CS proposes conditions and algorithms for the reconstruction of s-sparse codevectors of BSDR from much shorter noisy dense vectors stored in SDM if the linear transformation from the sparse codevectors to the stored dense vectors is known and obey Restricted Isometry Property (RIP).

2 CS-SDM MODEL

2.1 Construction

Consider the following design of s-sparse binary codevector storage CS-SDM. The storage consists of 3 units: encoder, memory (modified SDM) and decoder. The encoder converts input s-sparse codevectors conveniently for SDM dense form; the decoder converts the read data back in the s-sparse form. Similar storage architectures were proposed in the past (e.g. in [24]). The novelty of our approach is the construction of the units that implement the integration of the SDM memory and the CS decoder.

The encoder transforms M-dimensional sparse binary codevectors into m-dimensional integer vectors: $\boldsymbol{v} = \boldsymbol{\Lambda x}$. In our experiments, we used $m = k * s, \;\; k \in \{\, 8, \; 12 \,\}$.

The dictionary $\boldsymbol{\Lambda} \in \{\pm 1\}^{m \times M}$ is filled with pseudorandom uniformly distributed values and stored in regular memory. This matrix does not meet the condition of sample normalizing assumed by CS. But it is convenient for obtaining low amplitude integer values convenient for writing to the modified SDM. The required normalizing is postponed to the reading step and built in the decoder. Considering the normalization, the RIP property [24] is evident as the dot product of any two different samples of $\boldsymbol{\Lambda}$ is a Bernoulli distributed random variable with an expected value equal to 0; its division by the number of terms is equal to zero. That said, the samples are nearly orthogonal, which is an informal definition of RIP. Thus, CS conditions for the s-sparse codevector reconstruction are provided.

The memory unit is a slightly modified Jeackel's SDM. It includes the following minor changes:

- SDM contains integer counters in physical cells instead of conventional memory bits. When writing, SDM adds the written binary components to the counters (with 0 preliminary replaced by -1) in each cell activated by the address $\boldsymbol{a}$: the cell set is $A(\boldsymbol{a})$. Since the CS-SDM memory unit receives integers instead of binary values, they are simply added to the counters of the activated cells $\boldsymbol{u}^n$ without additional transformation:

$$u_i^n\,(t+1) = u_i^n\,(t) + v_i, \quad n \in A(\boldsymbol{a}) \;\; and \; i = \{1 \ldots m\} \tag{1}$$

- Each cell contains an additional (0^{th}) counter for further normalization. It is incremented during each writing to the cell:

$$u_0^n\,(t+1) = u_0^n\,(t) + 1, \quad n \in A(\boldsymbol{a}) \;\; and \; i = \{1 \ldots m\} \tag{2}$$

- The result of reading from the modified SDM is a real vector instead of a binary one. The reading is also simplified: there is no threshold. Just the normalized sum of all the activated cells is computed and submitted to the decoder:

$$v_i = \sum_{n \in A(\boldsymbol{a})} \frac{u_i^n}{u_0^n}, \quad i = \{1 \ldots m\} \tag{3}$$

The activation mechanism of the modified SDM unit is the same as of the conventional Jeackel's SDM. Each cell n contains a mask $\boldsymbol{t}^n$ of $K \ll M$ check bit numbers $1 \le t_k^n \le M$. The activation mechanism checks whether the address contains the values 1 in all the K positions recorded in the cell mask and, if so, activates the cell:

$$A(\boldsymbol{a}) \; = \left\{ n | r = t_k^n \; and \; a_r = 1 \; \forall k \in \{1, 2, \cdots, K\} \right\} \tag{4}$$

Here can be a question why aren't the addresses also compressed like the data by $\boldsymbol{\Lambda a}$? If they were compressed, the generalization property (the ability to correct minor errors) would be lost; even a single bit change in $\mathbf{a}$ would result in very different $\boldsymbol{\Lambda a}$ and activation of very different cells. So the addresses remain sparse in our construction. Besides, CS-SDM checks only the values 1 in addresses as they are supposed to be sparse.

It should be noted that in a computer implementation, CS-SDM may need longer counters than Kanerva or Jaeckel designs for the same number of recorded vectors. But the number of counters per cell is smaller: m instead of M.

Finally, the result read from the memory block is passed to the decoder, which searches for an s-sparse solution of the underdetermined system of linear equations by one of the CS algorithms. In the following tests, we evaluated decoding by linear programming [25] and CoSaMP [26]. The last is a greedy algorithm of the Orthogonal Matching Pursuit [27] family specially adapted for CS problems. Linear programming generally provides better solutions but requires significantly more computational time, so it has not been used for large tests. (Yet, we compared them in smaller tests.)

2.2 Testing and comparison

The associative memory mode was tested: writing and reading data via the address equal to the value. Half of the test data was made noisy: one 1 in the address was replaced by 0 when read. Clearing

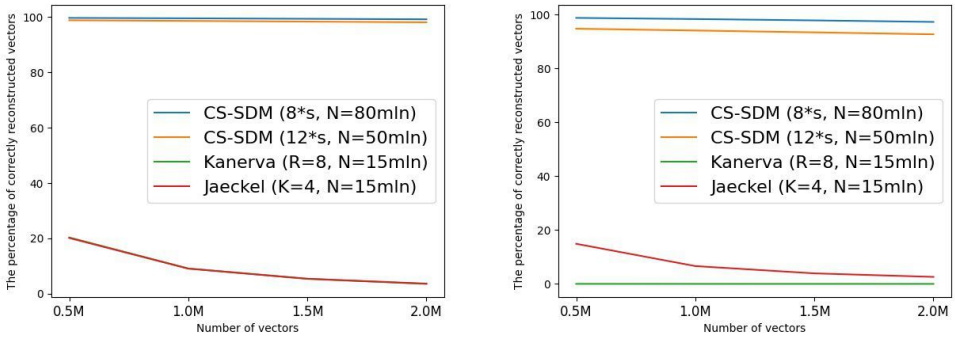

Figure 1: The percentage of correctly reconstructed vectors: exact address (left), address with single error (right), s=12

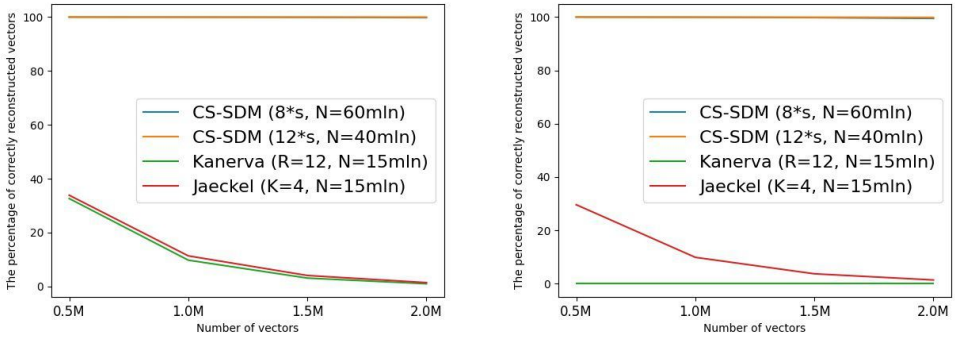

Figure 2: The percentage of correctly reconstructed vectors: exact address (left), address with single error (right), s=16

incomplete addresses is the main requirement for cleaning memory from the BSDR perspective.

We selected Jaeckel and Kanerva designs for comparison with CS-SDM; minor modifications were made in them too for better fitting the sparse nature of the test data. In our Jaeckel design, masks contain only the bit positions assuming activation if the address contains 1 in all the positions. It is precisely like in CS-SDM. For Kanerva's design, activation is caused by a small Hamming distance between the internal cell address and the submitted address. To account for the sparseness, the internal cell addresses were generated randomly, with exactly s 1 in each address.

2.3 Implementation

The described 3 SDM designs have been implemented on NVIDIA CUDA, except for the CS-SDM decoder, which uses existing CoSaMP implementation [28]. We also tested the LinProg procedure from the SciPy Python library (for comparison) [29, 30]. The test program source codes are open [31, 32].

3 EXPERIMENTS AND RESULTS

The experiments were run on NVIDIA GeForce RTX 2080Ti (Turing architecture, 11GB GDDR6 memory, 4352 CUDA kernels). We generated three datasets that contain random sparse binary vectors with the length $M = L = 600$ and sparsity $s = \{12, 16, 20\}$. The number of physical cells was selected so to fill the GPU memory. For CS-SDM, it was determined by the length of the input vectors: $m = k * s$ and varied between 30 and 100 million. For the reference designs, 15 million cells of $M = 600$ fitted in the memory.

The mask length $K = 4$ was used for both Jaeckel and CS-SDM designs. The value was selected w.r.t. the average number of activated cells. For the Kanerva model, the activation threshold was selected so that at least 4 features match is needed for a cell activation: $d = 2 * s - 4$.

Fig. 1-3 show that CS-SDM consistently provides almost 100% error-free recovery. On the other hand, Kanerva and Jaeckel's designs show a much worse recovery of the previously recorded data. The high frequency of their read errors can be partially explained by the insufficient number of physical cells because of less efficient memory use. However, the main reason seems to be the 0/1 probability asymmetry itself.

The experiments in Fig. 1–3 used a fast "greedy" CoSaMP algorithm to solve the CS problem. Tables 1-2 show how much the results could theoretically be improved with a more precise ℓ 1-minimization algorithm of linear programming. Due to the much longer computation time for LinProg, these experiments were performed for a much smaller number of codevectors: a hundred thousand. To make the difference more noticeable, we reduced the length of the physical cells: $k = \{6, 8\}$.

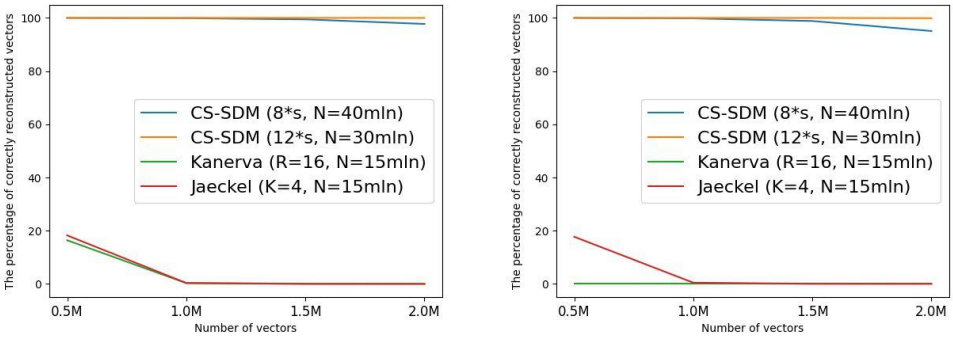

Figure 3: The percentage of correctly reconstructed vectors: exact address (left), address with single error (right), s=20

Table 1: Percentage of correctly reconstructed vectors for exact addresses

Number of features, $k = 6$			$k = 8$	
s	CoSaMP	LinProg	CoSaMP	LinProg
12	71.491 %	99.834 %	99.654 %	99.918 %
16	87.940 %	100 %	99.984 %	100 %
20	96.007 %	100 %	99.998 %	100 %

Table 2: Percentage of correctly reconstructed vectors for addresses with single error (0 instead of 1)

Number of features, $k = 6$			$k = 8$	
s	CoSaMP	LinProg	CoSaMP	LinProg
12	71.190 %	99.594 %	98.955 %	99.246 %
16	87.880 %	100 %	99.981 %	100 %
20	95.951 %	100 %	99.998 %	100 %

Tables 1-2 show the significant advantage of linear programming in the data reconstruction quality for short vectors, i.e. for high compression ratios m / M. However, with increased s and/or k when the length of the physical cell increases and the compression ratio decreases, the CoSaMP become as good as linear programming.

4 CONCLUSIONS

A new hybrid storage model CS-SDM based on integrating the classic Sparse Distributed Memory in the Vector Symbolic Architecture is proposed. CS-SDM demonstrates a significant advantage in the capacity and quality of cleaning memory for Binary Sparse Distributed Representations compared to the early SDM designs. The model efficiency is obtained due to the power of Compressive Sensing. CS-SDM brings together neural networks and the holistic symbolic representations of semantics.

From a conceptual human memory modelling perspective, whether the proposed design is relevant to the biological reality is essential. Both key elements – encoding via random matrix (interneuronal connections) and decoding via minimization (greedy algorithm) – seem primitive enough to be explorable in wildlife. Another open question is whether it is possible (and by what means) to modify this model to reduce the dependence on the homogeneity of the information stored in memory.

During the research, an open-source library [31, 32] of CS-SDM and two classical designs were implemented for the NVIDIA CUDA platform. The locality of data operations and homogeneity of SDM algorithms (fine-grained parallelism) make them easy portable on GPU and suitable for high-performance computing. GPU version of CoSaMP decoder was not implemented yet but can be developed. A complete GPU implementation not necessary for the study is desirable for further applications.

REFERENCES

[1] Pentti Kanerva. 1988. Sparse Distributed Memory. MIT Press, Cambridge, MA. 38-55.
[2] Michael J. Flynn., Pentti Kanerva, Neal Bhadkamkar. 1989. Sparse Distributed Memory: Principles and Operation. Report CSL-TR-89-400, Research Institute for Advanced Computer Science (RIACS), NASA Research Centre at Ames. 29-32.
[3] Pentti Kanerva. 1993. Sparse Distributed Memory and Related Models. Associative Neural Memories: Theory and Implementation. New York: Oxford University Press. 50-76.
[4] Lewis A. Jaeckel. 1989. An Alternative Design for a Sparse Distributed Memory. Report TR 89.28, Research Institute for Advanced Computer Science (RIACS), NASA Research Centre at Ames. 13-20.
[5] Lewis A. Jaeckel. 1989. A Class of Designs for a Sparse Distributed Memory. Report TR 89.30, Research Institute for Advanced Computer Science (RIACS), NASA Research Centre at Ames. 17-25.

[6] David A. Marr. 1969. Theory of Cerebellar Cortex. The Journal of Physiology. Vol. 202, N 2. Trinity College, Cambridge. 437-470.
[7] Derek J. Smith, Stephanie Forrest, Alan S. Perelson. 1998. Immunological memory is associative. Artificial Immune Systems and their Applications. Berlin: Springer. 105–112.
[8] Kenny Schlegel, Peer Neubert, Peter Protzel. 2020. A comparison of Vector Symbolic Architectures. arXiv:2001.11797v3 [cs.AI] https://arxiv.org/abs/2001.11797.
[9] Jerry A. Fodor, Zenon W. Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. Cognition. Vol. 28. 7–31.
[10] Paul Smolensky. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artificial Intelligence. Vol. 46, No 1–2. 159–216.
[11] Tony A. Plate. 1995. Holographic reduced representations. IEEE Transactions on Neural Networks. N 3. 41–59.
[12] Pentti Kanerva. 1994. The binary spatter code for encoding concepts at many levels.: Proc. of Intern. Conference on Artificial Neural Networks ICANN '94. Vol 1. London, Springer-Verlag. 226–229.
[13] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. Cognitive Computation. Vol. 1, N 2. 139–159.
[14] Gunnar Sjödin. 1998. The Sparchunk Code: A method to build higher-level structures in a sparsely encoded SDM. Proceedings of IEEE International Joint Conference on Neural Networks, IJCNN/WCCI'98. London: Springer. 50–58.
[15] Ross W. Gayler. 1998. Multiplicative binding, representation operators & analogy. Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences. New Bulgarian University, Sofia. 1–4.
[16] Dmitri A. Rachkovskij, Ernst M. Kussul. 2001. Binding and Normalization of Binary Sparse Distributed Representations by Context-Dependent Thinning. Neural Computation. Vol. 13, N 2. 411–452.
[17] Dmitri A. Rachkovskij. 2001. Representation and processing of structures with binary sparse distributed codes. IEEE Transactions on Knowledge and Data Engineering. Vol. 13, N 2. 261–276.
[18] Ross W. Gayler. 2003. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. Proc. ICCS/ASCS International Conference on Cognitive Science. Sydney, CogPrints, University of New South Wales. 133–138.
[19] E. Paxon Frady, Denis Kleyko, Friedrich T. Sommer. 2020. Variable Binding for Sparse Distributed Representations: Theory and Applications. arXiv:2009.06734v1 [cs.NE]. https://arxiv.org/abs/2009.06734
[20] Dmitri A. Rachkovskij. 2019. Codevectors: Sparse Binary Distributed Representations of Numerical Data (in Russian). Interservice, Kyiv, Ukraine. 200 p.
[21] Gunnar Sjödin. 1995. Convergence and new operations in SDM. SICS Research Report R95:13. Swedish Institute of Computer Science, Stockholm. 15 p.
[22] Emmanuel J. Candès, Justin Romberg, Terence Tao. 2006. Stable signal recovery from incomplete and inaccurate measurements. Comm. Pure Appl. Math. Vol. 59, N 8. 1207–1223.
[23] Emmanuel J. Candès, Michael B. Wakin. 2008. An introduction to compressive sampling. IEEE Signal Processing Magazine. Vol. 25, N 2. 21-30.
[24] Tiago Ramalho, Marta Garnelo. 2019. Adaptive posterior learning: few-shot learning with a surprise-based memory module. Proceedings of the 7th International Conference on Learning Representations (ICLR, New Orleans, Louisiana, USA). https://arxiv.org/abs/1902.02527
[25] George B. Dantzig. 1963. Linear programming and extensions. NJ, Princeton: Princeton University Press. 656.
[26] Deanna Needell, Joel A. Tropp. 2009. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. Applied and Computational Harmonic Analysis. Vol. 26, N 3. 301–321.
[27] Stephane Mallat, Zhifeng Zhang. 1993. Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing. Vol. 41, N 12. 3397–3415.
[28] Open-source library for CoSaMP algorithm: https://github.com/rfmiotto/CoSaMP/blob/master/cosamp.ipynb
[29] Paul Virtanen, Ralf Gommers, Travis E. Oliphant *et al.* 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods. Vol. 17, N 3. 261–272.
[30] Linear programming module from SciPy library: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html
[31] Ruslan O. Vdovychenko. 2021. The computer program "Hybrid neural memory model CS-SDM". Copyright #104882, May 26. Ukrainian Intellectual Property Institute.
[32] Open-source library CS-SDM: https://github.com/Rolandw0w/phd-sdm-cs.