Low-Power Sparse Hyperdimensional Encoder for Language Recognition

Mohsen Imani, John Hwang, and Tajana Rosing
University of California at San Diego

Abbas Rahimi and Jan M. Rabaey
University of California at Berkeley

Editor's note:
Online learning for data analysis, categorization and anomaly detection has become a key technique in a range of adaptive embedded applications. In this article the authors propose a very low power encoder design using a sparse, hyperdimensional representation of letters and words in natural language and they show that this representation and their design can be used with high energy efficiency for a language recognition task.

—Axel Jantsch, Royal Institute of Technology Electronics, Computer and Software Systems

Brain-inspired information processing architectures provide a promising new avenue to improve energy efficiency, asymptotically approaching the efficiency of brain computation, while tolerating variations in nanoscale fabrics [1]. In these architectures, patterns are used as basic data representation in contrast to computing with numbers. Hyperdimensional (HD) computing [2] captures and imitates this idea of distributed pattern-based data representations in the form of hypervectors-high-dimensional random vectors with dimensionality in thousands (e.g., D = 10,000 bits). HD computing builds upon a well-defined set of operations with random hypervectors and offers a complete computational paradigm that is easily applied to learning problems. Examples include analogy-based reasoning [3], sequence memory [4], language recognition [5]–[7], biosignal processing [8], and prediction from multimodal sensor fusion [9]. These applications typically use various encoding operations on dense hypervectors, where zeros and ones are equiprobable. These encoding operations, specifically bitwise XOR and shift, impose a large amount of switching activity that results in higher power consumption.

Digital Object Identifier 10.1109/MDAT.2017.2740839
Date of publication: 17 August 2017; date of current version: 14 November 2017.

In this paper, we propose a more biologically plausible [10], sparse binary representation, and related low-power encoding operations to produce and maintain sparse hypervectors where the number of ones is significantly less than zeros. The representation and encoding operations effectively control and reduce the large switching activity and power consumption of prior HD designs. We focus on an application of language recognition task for 21 European languages. During this task, the letters of an input sentence are projected to letter hypervectors that are combined to form n-gram, a sequence of n consecutive letters, hypervectors; these produced n-gram hypervectors are further superposed to construct a text hypervector for learning and classification. Our encoder projects each letter to a sparse letter hypervector of D-bit with an additional m-bit random *signature* where $m << D$. These signatures of n letters are used to generate a unique and sparse *n*-gram hypervector. The resulting *n*-gram hypervectors are superposed by a programmable thresholder that maintains the sparsity of produced text hypervector. We compare the efficiency and accuracy of our proposed sparse HD encoder with state-of-the-art HD designs using both dense [6] and sparse [4] codes. Our experimental evaluation shows that with the same level of accuracy, our sparse HD design provides 12.8× energy saving and 28.2× energy-delay product improvement.

Background and related work

Hypervectors are holographic and (pseudo) random with i.i.d. components. A hypervector contains all the information combined and spread across all its components in a full holistic representation so that no component is more responsible to store any piece of information than another. These unique features make a hypervector robust against errors in its components. Hypervectors can be manipulated with arithmetic operations, such as *binding* that forms a new hypervector which associates two hypervectors, and *bundling* that combines several hypervectors into a single composite hypervector. The reasoning in HD computing is based on similarity between the hypervectors. This similarity is measured by a distance metric.

In this paper, we target an application of HD computing for identifying the language of text samples, based on encoding consecutive letters into a hypervector. Recognizing the language of a given text is the first step in all sorts of language processing, such as text analysis, categorization, translation, etc. High-dimensional vector models are popular in natural-language processing and are used to capture meaning of a word from word-use statistics. Random indexing [11] is an algorithm based on high dimensionality and randomness and it provides a simple and scalable alternative to methods based on principal components, such as latent semantic analysis. We use random indexing for identifying the source language of text samples by generating and combining *n* consecutive letters (letter *n*-grams).

An HD architecture is proposed for recognizing a text language by generating and comparing text hypervectors [6]: the text hypervector of an unknown text sample is compared for similarity to precomputed text hypervectors of known language samples—the former is referred to as a query hypervector, while the latter are referred to as learned language hypervectors. The architecture has two main modules: encoding and associative memory. The encoding module projects an input text, composed of a stream of letters, to a hypervector. Then, this hypervector is broadcast to the associative memory module for comparing with a set of learned language hypervectors. The associative memory returns the language that has the closet match.

The hardware required by the associative memory to compute the Hamming distance essentially includes three primary components. These are an array of XOR gates to find dissimilar bits, a counter to count the number of mismatched bits in each row, and finally a comparator to find a row which has the minimum number of mismatches.

Although these dense representation and encoding operations can provide high classification accuracy [6], [8], they suffer from high power consumption due to the large amounts of switching activity imposed by the nature of dense representation. Other work aims to use sparsity in the hypervectors aligned with the direction in [4]; however, their sparse design achieves a lower classification accuracy on the language task compared to the dense design. In addition, their technique requires a large number of complex circular shifts within segments of the hypervector which wipes out the benefits of using sparse hypervectors. In contrast, we propose a novel sparse HD encoder that lowers the power consumption in *both operands and operators*. It maintains the sparsity of operands (hence lower switching activity) with simple and low-cost operators. Our HD encoder design also provides high classification accuracy comparable to the dense HD design, but significantly at lower power cost.

Sparse HD computing

Sparsity

In the dense HD design, hypervectors have equiprobable zeros and ones, i.e., 50% vector sparsity. When such hypervectors are combined using the dense encoding, it would result in a text hypervector with an expected sparsity of 50%. During test, an input text hypervector needs to be compared with all language hypervectors to find the most similar class. The bit-level comparison is done using an XOR array. During the similarity check, the number of switches in the associative memory is equal to the

Algorithm 1. Proposed sparse HD encoding

inputs: item hypervector (L), signature hypervector (S), training text size (H), n-gram size (N)
outputs: text/language hypervector (T)

```
 1: $iteration \leftarrow 0$
 2: for $c = 1 \ldots C_{language}$ do
 3:       for $i = 1 \ldots H_{text}$ do
 4:             for $k = 1 \ldots N_{gram}$ do
 5:                   $\rho_K \leftarrow K-1 + (S_N \oplus S_{N-1} \ldots \oplus S_{K+1} \oplus S_{K-1} \ldots \oplus S_1)$
 6:                   $Z_i^l \leftarrow Z_i^l + \rho_k(L_k^l)$
 7:             end for
 8:             $Z_{M_i}^l \leftarrow [Z_i^l]|_{THR=50\%}$
 9:             $T_c^l \leftarrow T_c^l + Z_{M_i}^l$
10:       end for
11:       $T_{M_c}^l \leftarrow [T_c^l]|_{THR}$
12:   end for
13:   $iteration \leftarrow iteration + 1$
```

number of bits different in those hypervectors. As dense HD design operates with random and orthogonal hypervectors, it can result in a maximum switching activity[1] of $\approx 100\%$ in the associative memory.

It is possible to reduce the switching activity by transitioning to a sparse hypervector representation. An important aim of the various algorithms used to encode item hypervectors is to make the representations of resulting language hypervectors consistent but different for each class. The dense design accomplishes this through the use of i.i.d. random hypervectors and combining n-gram hypervectors with a combination of shift and XOR operations [5], [6]. When the multiplication operation, which in the context of binary hypervectors is equivalent of the logical XOR operation, is applied to the sparse hypervectors, it does not bind the hypervectors effectively. Rather, when dealing with the sparse hypervectors, the XOR operation exhibits a behavior similar to that of an OR operation, which displays characteristics of its constituents. Due to the tendency of language to have bias toward similar n letter permutations, this characteristic of the XOR operation used in the dense HD encoding would find it difficult to differentiate similar n-grams. In addition to it, when using sparse hypervectors, the conventional dense encoding cannot maintain sparse n-gram hypervectors while obtaining the desired accuracy. In other words, the n-gram and text hypervectors can benefit far less from such initial sparsity. Thus, the associative memory will have a switching activity similar to that achieved when dense hypervectors are used.

[1]Switching activity here refers to switching probability at a gate output.

Proposed sparse HD encoder

Here, we propose a novel way to encode and bind sparse hypervectors to differentiate each language hypervector and give them unique representation. In our design, we assign a random and orthogonal hypervector with predefined sparsity to each letter in the alphabet. In sparse representation, the hypervectors are identified by the place of 1's rather than their holographic distribution. Therefore, the definition of orthogonality is different from that in dense representation, as in the sparse representation the orthogonal hypervectors need to have a dot product close to zero.

Our sparse encoding combines these hypervectors to create a unique hypervector representing each language. We propose an encoding which combines the sparse hypervectors, while keeping the combined hypervectors sparse and preserve the crucial information on each class of language. Algorithm 1 shows the steps followed by the proposed sparse encoder. After assigning the orthogonal hypervectors to each letter in a language, encoding starts by moving a window containing n letters (i.e., n-gram) through a text. Encoder combines hypervectors using a unique permutation on each n-gram index (lines 4–7) and by applying the thresholding function (line 8) to generate n-gram hypervectors. Finally, the algorithm accumulates n-gram hypervectors and applies the same thresholding function using a custom threshold (*THR*) value (line 11).

Going into details of the proposed encoding, our technique assigns a random m-bit signature (S) to each sparse letter hypervector. The goal of these signatures is to create a consistent but unique shift for each letter hypervector while generating the n-gram hypervector. This shift enables not only our design to differentiate between the combination of different letters but also the order of their presence in n-gram. Therefore, to bind hypervectors together in an n-gram, each hypervector is shifted by the XOR of the m-signature bits of all other letters in the n-gram window. This operation in effect applies a unique shift in the range of $0 \ldots 2^m - 1$ to each letter hypervector. The actual number of shifts to each item hypervector can be obtained by

$$\rho_K = K - 1 + (S_N \oplus S_{N-1} \ldots \oplus S_{K+1} \oplus S_{K-1} \ldots \oplus S_1)$$

where ρ_K shows the number of shifts (or permutations) required for kth letter in the n-gram and S_h shows the signature of the hth letter in the n-gram.The sparse hypervectors corresponding to the letters in the n-gram ($L_1^l, L_2^l, L_3^l, \ldots, L_N^l$) are combined as follows to generate an n-gram hypervector:

$$Z_{M_i}^{l} = \left[\rho_N\left(L_N^l\right) + \rho_N - 1\left(L_{N-1}^l\right) + \ldots + \rho_1\left(L_1^l\right)\right]\Big|_{THR=50\%}$$

where $[+]\,|_{THR}$ is the thresholding function which looks at the value at each index, and in the case of surpassing a certain *THR* level, it is represented by "1." Finally, in order to generate text/language hypervectors, the n-grams ($Z_{M_i}^l$) are combined using another thresholding function but with a controllable *THR* value (line 11).

$$T_{M_c}^{l} = \left[Z_{M_1}^l + Z_{M_2}^l + \ldots + Z_{M_H}^l\right]\Big|_{THR}$$

where T_M is the generated text hypervector. The primary purpose of the thresholding function is to apply thinning to the n-gram hypervectors for maintaining an appropriate density.

Sparse encoding parameters

Here, we explore the impact of different parameters on the accuracy and efficiency of the proposed sparse encoding. For optimal results, m should be large enough to create a unique signature for each letter ($2^m > N_{letters}$). Alternatively, at the cost of a slight decrease in the accuracy, some letters can use the same signature in the case where maintaining a small m is important. Increasing the value of m improves the classification accuracy by assigning a unique signature to each letter. However, our results show that HD accuracy starts saturating for m larger than 5 bits, because using 5 bits is enough to provide a unique signature for each letter in the EU languages. In terms of energy consumption, using large m slightly degrades the energy efficiency of HD computing by increasing the number of the required shift operations. For sparse design, we set m to 4 bits to provide the maximum accuracy of 95.4%.

The thresholding function sets the sparsity of text hypervector by controlling the value of *THR*. *THR* has impact on the accuracy and efficiency of language classification. Our design exploits this parameter as a means of setting the sparsity of the text/language hypervectors. Table 1 shows the impact of using different *THR* values on the classification accuracy, text hypervector sparsity, and the switching activity of the proposed HD design. The HD energy and switching activity increase with the *THR* value, as large *THR* results in higher switching activity in the associative memory. In terms of HD accuracy, *THR* = 30% provides maximum classification accuracy. Using smaller *THR* significantly increases the text hypervector sparsity and results in missing information. On the other hand, large *THR* value, i.e., 40%–50%, increases the density of text hypervectors and results in lower classification accuracy. In this work, we set the *THR* = 30% in order to provide the maximum classification accuracy with about 6.1% text hypervector sparsity and 11.1% switching activity.

Hardware implementation

Figure 1 shows hardware implementation of the proposed sparse design, consisting of both the encoder and associative memory. The encoder block generates the signature (m-bit) and sparse letter hypervector (D-bit) for every input letter using the item memory. The letter hypervectors and related signatures are fetched in their order in which they are present in the text. Based on a signature, first our design generates a unique shift for each letter hypervector in the n-gram window (n = 3 in Figure 1). Then, the encoder shifts the letter hypervectors based on these shift values. These shifted hypervectors are combined using accumulator and thresholding technique to generate the first n-gram hypervector. For the second iteration, a signature and a letter hypervector of the next letter are fetched to the window (in a FIFO fashion) to generate the second n-gram hypervector. For example, if the input text is "HELLO," the "H," "E," and "L" are the letter hypervectors 1 to 3, respectively, to generate the first n-gram. For the second iteration, "E," "L," and "L" are

Table 1. Impact of THR value on the classification accuracy, text hypervector sparsity, and total switching activity of the proposed sparse HD design.

THR **value**	**50%**	**40%**	**30%**	**20%**	**15%**	**10%**
Accuracy	91.7%	94.2%	95.4%	94.7%	93.9%	79.1%
Text Vector Sparsity	8.6%	7.6%	6.1%	4.9%	4.2%	1.8%
Switching Activity	12.8%	11.1%	9.5%	8.1%	7.5%	3.5%

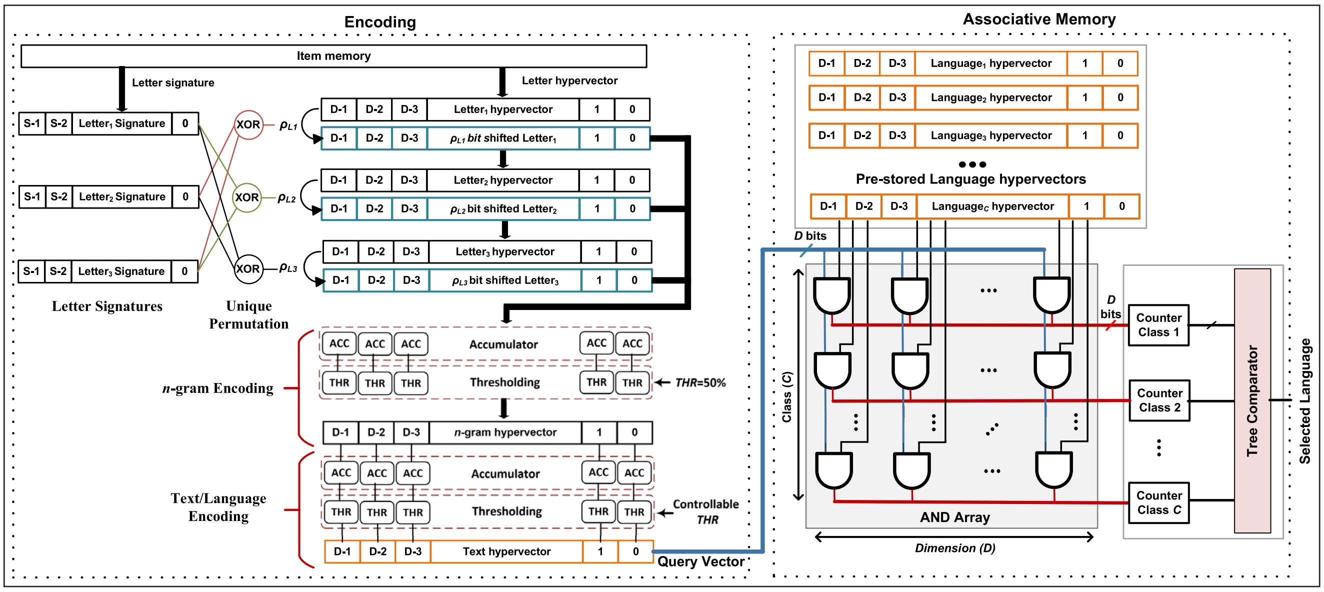

Figure 1. The proposed sparse HD implementation: (a) encoding block and (b) associative memory.

placed in letter hypervectors 1 to 3, respectively, to generate the second *n*-gram, and so on. After going through the entire text input, an accumulator and a thresholding block generate the text hypervector by combining all the produced *n*-gram hypervectors. Figure 1b shows a digital hardware design of the associative memory consisting of three stages: AND array, counter, and comparator.

The associative memory checks similarity in the test mode. While the prior dense design uses Hamming distance or cosine similarity as a metric to find the most similar row, the proposed sparse design makes the hypervector comparisons using dot product. This is because of the importance given to the position of the few 1's in each hypervector. In the sparse design, the goal is to calculate the number of indices with aligned ones, which is more efficiently accomplished using dot product metric or its binary equivalent, the AND operation. Therefore, our design uses an array of AND gates (instead of XOR gates in the dense associative memory) to calculate the similarity of the input query hypervector to all stored language hypervectors. This is a major advantage in terms of energy consumption, as AND gate array not only requires fewer transistors to implement, but also involves significantly lower switching activity in comparison to XOR due to its asymmetric output toward a 0.

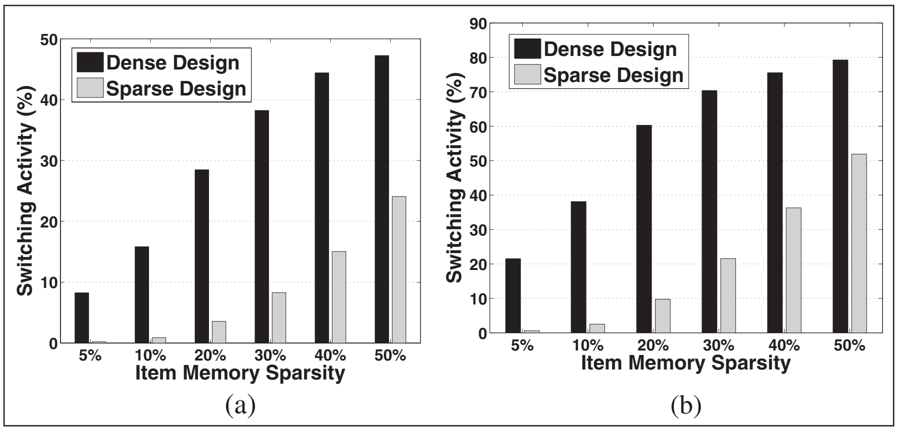

Figure 2. Comparison of the switching activity of dense and proposed sparse HD in (a) encoding and (b) associative memory blocks for the different values of input hypervector sparsity.

In the second stage, a counter calculates bit similarity using the output of the AND array. Finally a comparator block, implemented in a tree structure, searches the counter outputs to find a class which has the maximum similarity to the input query. In the prior dense design [6], the text hypervectors are expected to be dense with 50% elements as 1. These dense hypervectors can incur the maximum switching activity of 100% during the associative search. Figure 2 shows the switching activity of the encoding and associative memory blocks when the input sparsity changes from 5% to 50%. In the dense

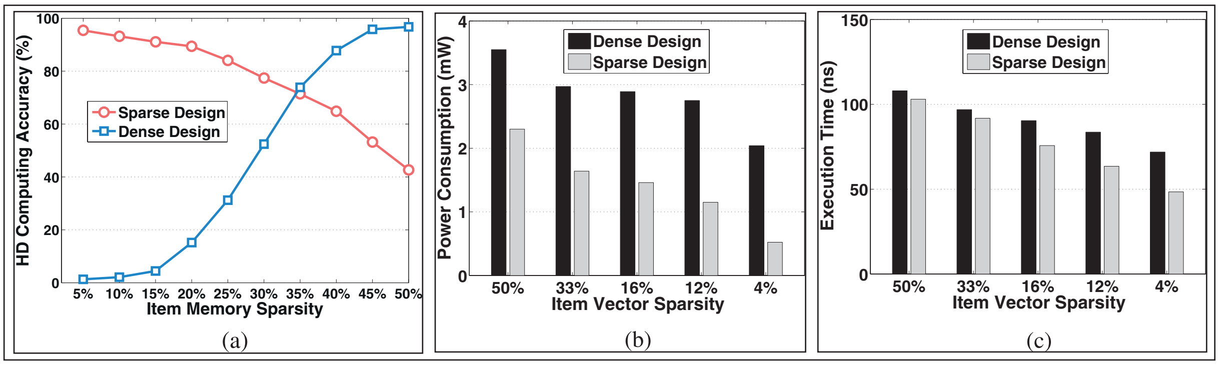

Figure 3. The energy consumption and execution time of the dense and the sparse HD designs when item memory uses different values of hypervector sparsity. (a) Accuracy. (b) Energy consumption. (c) Execution time.

design, the sparsity can result in lower switching activity mostly in the encoding block. The way in which the dense HD design encodes the item hypervectors results in generating *n*-gram and text hypervectors with dense representation, regardless of their initial sparsity in item memory. This eliminates the possibility of benefiting from hypervector sparsity in the associative memory of the dense design. In contrast, the sparse encoding tries to keep the sparsity of the text hypervectors low using the thresholding function, to reduce the switching activity of both encoder and associative memory. Another aspect of the design affected by this feature is the maximum size required for the counter and comparator. While the dense design may require counters as large as the logarithm of dimension (*D*), the size of counter in sparse design is bounded by the lowest sparsity of the trained language hypervectors (i.e., the maximum number of ones). The lowest language hypervector sparsity depends on the *THR* value in encoding scheme. We discuss more about the counter and comparator size in the following section.

Evaluations

Experimental setup

We compare the power, execution time, and accuracy of the dense and proposed sparse HD designs. We describe these designs in a parametrized manner using RTL *SystemVerilog*. For the synthesis, we use *Synopsys Design Compiler* with the TSMC 45-nm technology library for the general purpose process with high V_{TH} cells. We extract the switching activity using ModelSim by applying the test sentences. We measure the power consumptions using *Synopsys PrimeTime* at (1 V, 25 °C, and TT) corner.

To test the efficiency, we apply the application to the recognition of 21 European languages. The training and testing data sets are taken from the Wortschatz Corpora [12]. The training texts for the language hypervectors are around one megabyte in size. We use 1,000 testing sentences that are taken from the Europarl Parallel Corpus [12]. The accuracy represents the percentage of correctly recognized sentences from 21,000 test samples. The accuracy of the various designs is tested through Matlab simulations.

Accuracy

Figure 3a shows the accuracy of the dense and the sparse designs for different values of item memory sparsity. As the figure shows, these two designs exhibit completely different sensitivity to sparsity. In the sparse design, the shift operation has a significant impact on differentiating two different hypervectors. This effect is not significant for hypervectors of greater density. The difference in our sparse and the previous dense approaches is that our sparse design is more about patterns in positions of ones, while the dense design is more about establishing a very distributed series of patterns. In the dense design, the accuracy significantly drops with the reducing sparsity of the *n*-gram hypervectors to a degree which wipes out the benefits provided by the sparse seed hypervectors. In contrast, the proposed sparse HD design has significantly a higher accuracy at a higher sparsity. This can be credited to the proposed encoding which makes the hypervectors unique.

Sparsity and efficiency

In terms of energy consumption, the proposed sparse HD has noticeably better performance than the dense HD designs due to the following.

Table 2. Classification accuracy of the proposed sparse and the dense HD designs with a different dimensionality (D).

	n-gram	2000 bits	4000 bits	6000 bits	8000 bits	10,000 bits
sparse	4	74.18%	86.5%	91.5%	93.6%	95.4%
dense	3	69.4%	86.4%	90.5%	92.1%	96.1%

- Using the dot product instead of Hamming distance metric to find the hypervector with closest similarity. This is the logical equivalent of using an efficient `AND` array rather than an energy hungry `XOR` array to determine the hypervector similarity. Our evaluation shows that the energy consumption of an `AND` array in the context of our application is around 60% of that of the `XOR`, in addition to its lower delay.
- The switching activity of the bit-level comparison significantly decreases using the sparse hypervector. Figure 3b shows the power consumption of the dense and proposed sparse HD designs when the input sparsity varies from 4% to 50%. In each configuration, we set the HD parameters (i.e., the *n*-gram size and *THR* value) such that we achieve the best energy improvement. The result shows that the noticeable energy benefit of the sparse HD is caused by its higher sparsity rather than replacing the `XOR` with `AND` gates.

Figure 3c also compares the execution time of HD designs in a different sparsity. The result shows that for the same sparsity, the proposed sparse design works faster than the dense design. This improvement is partly due to using the dot product similarity implemented using a fast `AND` array to compute the most similar hypervector. However, the main sparse design speedup benefit comes from its smaller counter and comparator blocks. As we discussed in hardware implementation the dense design requires counter to support up to *D*-bit counting, while in the sparse design this number is much lower. In the sparse design, the highest sparsity of the trained hypervectors, usually about 6%, determines the maximum bitwidth of the counter and comparator blocks. Our evaluations show that while achieving the maximum accuracy of 95%, the proposed sparse design can provide 12.8× energy reduction, 2.2× speedup, and 28.2× energy-delay product improvement in comparison to the dense design.

Table 2 shows the impact of dimension scalability on the classification accuracy of dense and sparse HD. The results show that both HD designs have high robustness to dimension reduction, however, the sparse HD shows higher robustness specially when the dimensionality goes below 4000 bits.

Table 3. Energy consumption, execution time, and sparsity of the segmented and our proposed sparse design for maximum accuracy (D = 10,000 bits).

	Accuracy	Sparsity	Energy	Exe. Time
Proposed sparse	95.4%	4%	4.0*pJ*	48.4*ns*
Segmented [4]	92.6%	10%	21.2*pJ*	68.7*ns*

Area efficiency comparison

Here, we compare the area of different HD designs. For each design, the area depends on

- the number of transistors involved in the implementation of the comparison array;
- the maximum size of counter and comparator required to count and compare the hypervectors. The estimation made by *Synopses Design Compiler* shows that the proposed sparse design has 34% lower area compared to the dense design.

Finally, we compare the efficiency of the proposed design with the prior sparse HD design, segmented, proposed in [4]. To achieve the maximum accuracy, our sparse and segmented designs use the sparsity of 4% and 10%, respectively. As Table 3 shows, the maximum accuracy that segmented can achieve is 92.6%, which is 2.8% less than our proposed sparse design. In addition, while considering the efficiency, our proposed design can provide 5.2× energy savings and 1.4× speedup (7.28× EDP improvement) as compared to the segmented design. This efficiency is due to higher sparsity and simpler encoding scheme used in our proposed sparse design.

Fast and one-shot learning capabilities of brain-inspired HD computing make it a prime candidate for on-chip learning. However, the large amount of switching activities, imposed by dense binary hypervectors, increases the power consumption. We propose a more biologically plausible sparse binary representation where the number of ones is significantly less than zeros in the hypervectors.

We design a low-power encoder which partitions an input sentence into a set of letter *n*-grams and computes the related *n*-gram hypervectors and text hypervectors while maintaining the sparsity. We show its application for European language recognition task. Our experimental evaluation shows that, for the same level of classification accuracy, our sparse HD design provides 12.8× energy reduction and 28.2× energy-delay product improvement as compared to the state-of-the-art dense HD design. ■

Acknowledgments

This work was supported by Systems on Nanoscale Information fabriCs, one of the six SRC STARnet Centers, sponsored by MARCO, DARPA, and NSF Grant 1527034.

■ References

[1] H. Li et al., "Hyperdimensional computing with 3d VRRAM in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *Proc. IEEE Int. Electron Devices Meeting,* 2016, pp. 16–20.

[2] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognit. Comput.,* vol. 1, no. 2, pp. 139–159, 2009.

[3] P. Kanerva, "What we mean when we say "whats the dollar of mexico?": Prototypes and mapping in concept space," in *Proc. AAAI Fall Symp.: Quantum Informatics for Cognitive, Social, and Semantic Processes,* 2010, pp. 2–6.

[4] M. Laiho, J. H. Poikonen, P. Kanerva, and E. Lehtonen, "High dimensional computing with sparse vectors," in *Proc. Biomed. Circuits Syst. Conf.,* 2015, pp. 1–4.

[5] A. Joshi, J. Halseth, and P. Kanerva, "Language geometry using random indexing," in *Proc. Quantum Interaction 2016 Conf.,* 2016, pp. 265–274.

[6] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy efficient classifier using brain-inspired hyperdimensional computing," in *Proc. 2016 IEEE/ACM Int. Symp. Low Power Electronics Des.,* Aug. 2016, pp. 64–69.

[7] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *Proc. 2017 IEEE Int. Symp. High Performance Comput. Architecture,* 2017, pp. 445–456.

[8] A. Rahimi, P. K. L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition," in *Proc. IEEE Int. Conf. Rebooting Comput.,* Oct. 2016.

[9] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 27, no. 9, pp. 1878–1889, 2016.

[10] A. Litwin-Kumar, K. D. Harris, R. Axel, H. Sompolinsky, and L. F. Abbott, "Optimal degrees of synaptic connectivity," *Neuron,* vol. 93, pp. 1153–1164.e7, May 2017.

[11] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proc. 22nd Annu. Conf. Cognit. Sci. Society,* 2000, vol. 1036, Citeseer, pp. 1–2.

[12] U. Quasthoff, M. Richter, and C. Biemann, "Corpus portal for search in monolingual corpora," in *Proc. 5th Int. Conf. Language Resources Evaluation,* 2006, p. 21.

Mohsen Imani is a member of the System Energy Efficiency Laboratory and is currently with the University of California (UC) Berkeley's Wireless Research Center on neuromorphic computing projects. He is a PhD candidate at the Department of Computer Science and Engineering, UC San Diego. His research interests are approximate computing, brain-inspired computing, and processing-in-memory architecture.

John Hwang is an Engineer at Parsons Corporation Inc. He received a BS from the Department of Computer Science and Engineering, UC San Diego.

Tajana Rosing is a Professor, a holder of the Fratamico Endowed Chair, and the Director of the System Energy Efficiency Laboratory at the University of California San Diego. She is currently heading the effort in SmartCities as a part of DARPA and industry-funded TerraSwarm Center. Her research interests are energy efficient computing and embedded and distributed systems.

Abbas Rahimi is currently a Post-Doctoral Scholar at the Department of Electrical Engineering and Computer Sciences, University of California (UC) Berkeley, CA, USA. He received a PhD in computer science and engineering from UC San Diego, CA, USA, in 2015. He is a member of the Berkeley Wireless Research Center.

Jan M. Rabaey holds the Donald O. Pederson Distinguished Professorship at the University of California at Berkeley. He is a Founding Director of the Berkeley Wireless Research Center and the Berkeley Ubiquitous Swarm Lab and is currently the Electrical Engineering Division Chair at Berkeley.

■ Direct questions and comments about this article to Mohsen Imani, Department of Computer Science and Engineering, University of California at San Diego, CA 92093 USA; e-mail: moimani@ucsd.edu.