High-Dimensional Computing as a Nanoscalable Paradigm

Abbas Rahimi, *Member, IEEE*, Sohum Datta, *Student Member, IEEE*, Denis Kleyko, Edward Paxon Frady, *Member, IEEE*, Bruno Olshausen, Pentti Kanerva, and Jan M. Rabaey, *Fellow, IEEE*

***Abstract*— We outline a model of computing with high-dimensional (HD) vectors—where the dimensionality is in the thousands. It is built on ideas from traditional (symbolic) computing and artificial neural nets/deep learning, and complements them with ideas from probability theory, statistics, and abstract algebra. Key properties of HD computing include a well-defined set of arithmetic operations on vectors, generality, scalability, robustness, fast learning, and ubiquitous parallel operation, making it possible to develop efficient algorithms for large-scale real-world tasks. We present a 2-D architecture and demonstrate its functionality with examples from text analysis, pattern recognition, and biosignal processing, while achieving high levels of classification accuracy (close to or above conventional machine-learning methods), energy efficiency, and robustness with simple algorithms that learn fast. HD computing is ideally suited for 3-D nanometer circuit technology, vastly increasing circuit density and energy efficiency, and paving a way to systems capable of advanced cognitive tasks.**

***Index Terms*— Alternative computing, bio-inspired computing, hyperdimensional computing, vector symbolic architectures, in-memory computing, 3D RRAM, pattern recognition.**

I. Introduction

OVER the past six decades, the semiconductor industry has been immensely successful in providing exponentially increasing computational power at an ever-reducing cost and energy footprint. Underlying this staggering evolution is a set of well-defined abstraction layers: starting from robust switching devices that support a deterministic Boolean algebra, to a scalable and stored program architecture that is Turing complete and hence capable of tackling (almost) any computational challenge. Unfortunately, this abstraction chain is being challenged as scaling continues to nanometer dimensions, as well as by exciting new applications that must support a myriad of new data types. Maintaining the current deterministic computational model ultimately puts a lower bound on the energy scaling that can be obtained, set in place by fundamental physics that governs the operation, variability and reliability of the underlying nanoscale devices [1]. These pose a computational challenge for the Internet of Things (IoT) that require massive amounts of local processing [2].

At the same time, it is clear that the nature of computing itself is evolving rapidly. For a vast number of IoT applications, cognitive functions such as classification, recognition, synthesis, decision-making, and learning are rapidly gaining importance in a world that is infused with sensing modalities and in need of efficient information-extraction. This is in sharp contrast to the past when the central objective of computing was to perform calculations on numbers and produce results with extreme numerical accuracy.

Indeed, the recent success of deep-learning networks – based on the artificial neural networks of the past – is based on finding ever expanding applications from speech and image recognition, to predicting the effects of mutations in noncoding DNA on gene expression and disease. While these success stories foretell an intriguing future for learning machines, the current reality is that these approaches need to run a large cluster of modern computers for several days in order to perform the required computations. In doing so, they also consume gigantic amounts of energy. Thus, the combination of the digital computer and deep-learning algorithms, while very important as a proof of concept, is neither realistic nor scalable for the broad societal adoption. Therefore, realizing the full potential of learning machines requires significant improvement in every aspect of the hierarchy of computers.

Conventional deep-learning algorithms require brute force tuning of millions of parameters over repetitive iterations at every layer of a several-layer stack. As a result, learning is slow and power hungry – every weight change during the training expends energy. Shuttling data through these networks during testing/production is also power hungry. Currently, there are no guarantees regarding the optimality or efficiency of operations in these networks. It is, therefore, essential to find new algorithms which are capable of "online" or one-shot learning, and for which there exist computational theories that bound the consumption of resources and the computational complexity for a given task; see Section II-B1. At the same time, significant technological advances are required to create new physical devices that will form the building blocks of future learning machines, so that the energy costs can be substantially lowered. To achieve optimum energy usage, these devices must be integrated and organized in efficient architectures that are tuned to the intricacies of the corresponding

Manuscript received December 7, 2016; revised April 13, 2017; accepted May 8, 2017. Date of publication June 7, 2017; date of current version August 28, 2017. This work was supported by the Systems on Nanoscale Information fabriCs, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA. This paper was recommended by Associate Editor A. Sangiovanni Vincentelli. *(Corresponding author: Abbas Rahimi.)*

A. Rahimi, S. Datta, and J. M. Rabaey are with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: abbas@eecs.berkeley.edu; sohumdatta@eecs.berkeley.edu; jan@eecs.berkeley.edu).

D. Kleyko is with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden (e-mail: denis.kleyko@ltu.se).

E. P. Frady, B. Olshausen, and P. Kanerva are with the Helen Wills Neuroscience Institute, University of California at Berkeley, Berkeley, CA 94720 USA.

Digital Object Identifier 10.1109/TCSI.2017.2705051

learning algorithms. In order to continue miniaturization of the component base, new architectures will need to utilize the"intrinsic" properties of the underlying nano-devices, such as embracing nanoscale device variability to meet application needs instead of paying high energy costs to avoid variability.

Finally, we are challenged positively by biology. Brains receive a steady stream of input from hundreds of millions of sensory neurons, process it with neurons that are stochastic and slow (10-to-100 Hz compared to computer circuits operating at speeds a million times faster), consume very little energy (roughly equivalent to a laptop computer), and produce behavior unparalleled by computers. To allow us to function in the world, our brains extract information from vast quantities of noisy data.

What brains have in their favor are numbers. The human cerebellum alone, with its wiring resembling computer RAM's, has on the order of 3 trillion modifiable synapses. If learning were nothing more than the setting up a lookup table and if a synapse were the equivalent of one bit, the cerebellum could store 10 million bytes of data per day for 100 years, or about a thousand bits per second, day and night. There must be models of computing rather different from the present and more akin to brains, that benefit from such large numbers and cope with a profusion of real-world data in real time.

Recognizing these needs, we propose an energy-efficient designs for a theory of high-dimensional computing (HD computing), also referred to as "hyperdimensional" on account of a dimensionality that is in the thousands [3]. In this formalism, information is represented in ultra high-dimensional vectors (hypervectors). Such hypervectors can then be mathematically manipulated to not only classify but also to make associations, form hierarchies, and perform other types of cognitive computations. In addition, the set of mathematical operations on hypervectors enables capabilities beyond deep neural networks such as one-shot learning and short-term (working) memory. Thus, HD computing can substantially reduce the number of operations needed to perform cognitive functions compared to conventional deep-learning algorithms, thereby providing tremendous energy savings.

This article presents an overview of the HD computing paradigm and its application in several computational tasks. We outline its implementation in large arrays of nonvolatile memory. Nonvolatile memory such as Resistive RAM can be integrated at high density with logic switches. Integrating in 3D will allow in-memory execution of all the required HD vector operations, resulting in significant energy savings as there is no need to shuttle data back and forth between processing and memory units. Combined with the unique properties of HD computing, we expect to see learning machines that are orders of magnitude more efficient than the ones in use today.

This paper is organized as follows. In Section II, we introduce HD computing and discuss its key properties. In Section III we present a 2D hardware architecture for HD computing and describes how its arithmetical operations can be used to solve a classification problem. Our experimental results for the 2D architecture regarding robustness, energy efficiency and information capacity are described in Section IV. In Section V and Section VI, we project a 3D architecture and potential algorithms and application drivers for HD computing, and conclude in Section VII.

II. High-Dimensional Computing

The difference between traditional computing and high-dimensional computing is apparent in the elements that the computer computes with. In traditional computing the elements are Booleans, numbers, and memory pointers. In HD computing they are multicomponent vectors, or tuples, where neither individual component nor a subset thereof has a specific meaning: a component of a vector and the entire vector represent the same thing. Furthermore, the vectors are wide: the number of components is in the thousands.

We will demonstrate the idea with a simple example from language [4]. The task is to identify the language of a sentence from its three-letter sequences called *trigrams*. We compare the trigram profile of the sentence to the trigram profiles of 21 languages and chose the language with the most similar profile. A profile is essentially a histogram of trigram frequencies in the text in question.

The standard algorithm for computing the profile – the *baseline* – scans through the text and counts the trigrams. The Latin alphabet of 26 letters and the space give rise to $27^3 = 19{,}683$ possible trigrams, and so we can accumulate the trigram counts into a 19,683-dimensional vector and compare such vectors to find the language with the most similar profile. This is straightforward and simple with trigrams but it gets complicated with higher-order n-grams when the number of possible n-grams grows into the millions (the number of possible pentagrams is $27^5 = 14{,}348{,}907$). The standard algorithm generalizes poorly.

The HD algorithm starts by choosing a set of 27 letter vectors at random. They serve as *seed vectors,* and the same seeds are used with all training and test data. We have used 10,000-dimensional vectors of equally probable 1s and -1s as seeds. From these we make trigram vectors by *rotating* the first letter vector twice, the second letter vector once, and use the third letter vector as is, and then by *multiplying* the three vectors component by component. Such trigram vectors resemble the seed vectors in that they are 10,000-D with equally probable 1s and -1s, and they are random relative to each other. A text's profile is then the sum of all the trigrams in the text: for each occurrence of a trigram in the text, we *add* its vector into the profile vector. The profile of a test sentence is then compared to the language profiles and the most similar one is returned as the system's answer, as above. In contrast to the standard algorithm, the HD algorithm generalizes readily to any n-gram size – the HD vectors remain 10,000-D. This example will be analyzed further in Section III on 2D Architecture for HD Computing.

A. Distributed Representations and Arithmetic Operations on Hypervectors

HD computing is based on the properties of high-dimensional vectors and operations on them. We will review them with reference to D-bit vectors, where $D = 10{,}000$

TABLE I

SUMMARY OF HD COMPUTING FRAMEWORKS. EACH FRAMEWORK HAS ITS OWN SET OF SYMBOLS AND OPERATIONS ON THEM FOR ADDITION, MULTIPLICATION, PERMUTATION, AND A MEASURE OF SIMILARITY

VSAs	Ref.	Symbol Set	Multiplication	Addition	Permutation	Similarity metric
BSC	[6]	dense binary vectors	elementwise XOR	majority function	rotation	Hamming
MAP	[7]	dense bipolar vectors	elementwise multiplication	elementwise addition	rotation	cosine
HRR	[12]	unit vectors	circular convolution	elementwise addition	none	dot product
FHRR	[5]	complex unitary vectors	elementwise multiplication	angle of sum	circular convolution	cosine
MBAT	[9]	dense bipolar vectors	vector-matrix multiplication	elementwise addition	multiple binding	dot product
BSDC	[8]	sparse binary vectors	context-dependent thinning	elementwise disjunction	rotation	overlap of vectors
GAHRR	[10]	unit vectors	geometric product	elementwise addition	none	unitary-space scalar product

for example. There are 2^D such vectors, also called *points*, and they correspond to the corners of a D-dimensional unit cube. The number of places at which two binary vectors differ is called the *Hamming distance* and it provides a measure of *similarity* between vectors. A peculiar property of high-dimensional spaces is that most points are relatively far from any given point. Hence two D-bit vectors chosen at random are dissimilar with near certainty: when referenced from the center of the cube they are nearly *orthogonal* to each other.

HD computing uses three operations [3]: *addition* (which can be weighted), *multiplication*, and *permutation* (more generally, multiplication by a matrix). "Addition" and "multiplication" are meant in the abstract algebra sense where the sum of binary vectors $[A + B + \ldots]$ is defined as the componentwise majority with ties broken at random, the product is defined as the componentwise Exclusive-Or (addition modulo 2, $\oplus$), and permutation (ρ) shuffles the components. All these operations produce a D-bit vector.

The usefulness of HD computing comes from the nature of the operations. Specifically, addition produces a vector that is *similar* to the argument vectors – the inputs – whereas multiplication and random permutation produce a *dissimilar* vector; multiplication and permutation are *invertible*, addition is approximately invertible; multiplication *distributes* over addition; permutation distributes over both multiplication and addition; multiplication and permutation *preserve similarity*, meaning that two similar vectors are mapped to equally similar vectors elsewhere in the space.

Operations on HD vector can produce results that are approximate or "noisy" and need to be identified with the exact vectors. For that, we maintain a list of known (noise-free) seed vectors, called *item memory* or *clean-up memory.* When presented with a noisy vector, the item memory outputs the most similar stored vector. High dimensionality is crucial to make that work reliably. With 10,000-bit vectors, 1/3 of the bits can be flipped at random and the resulting vector can still be identified with the original stored one.

The operations make it possible to encode and manipulate sets, sequences and lists – in essence, any data structure. We show how a data record consisting of variables x, y, z with values a, b, c can be encoded into a hypervector H and the value of x can be extracted from it. We start with randomly chosen seed vectors X, Y, Z, A, B, C for the variable and the values and store them in the item memory. We then encode the record by *binding* the variables to their values with multiplication and by adding together the bound pairs:

$$H = [(X \oplus A) + (Y \oplus B) + (Z \oplus C)]$$

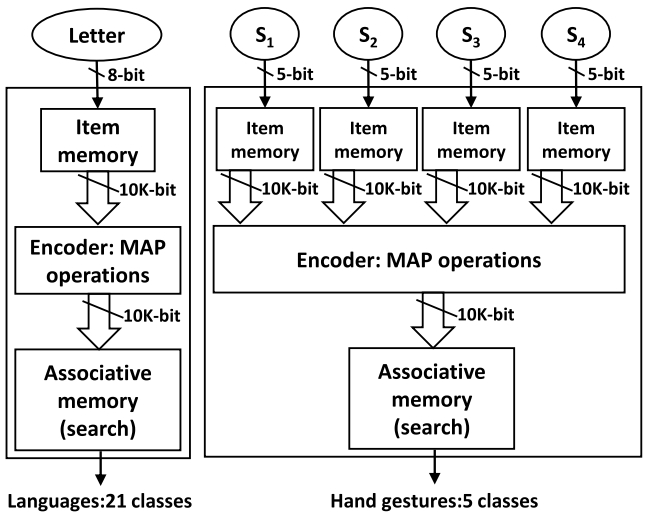
To find the value of x in H we multiply it with the inverse of X, which for XOR is X itself: $A' = X \oplus H$. The resulting vector A' is given to the item memory which returns A as the most-similar stored vector. An analysis of this example would show how the properties of the operations, as listed above, come to play. A thing to note about the operations is that addition and multiplication approximate an algebraic structure called a field, to which permutation gives further expressive power.

HD computing has been described above in terms of binary hypervectors (the bipolar vectors of the first example are equivalent to the binary). However, the key properties are shared by high-dimensional vectors of many kinds, all of which can serve as the computational infrastructure. They include Holographic Reduced Representations (HRR) [5], frequency domain Holographic Reduced Representations (FHRR) [5], Binary Spatter Codes (BSC) [6], Multiply-Add-Permute (MAP) architecture [7], Binary Sparse Distributed Codes (BSDC) [8], Matrix Binding of Additive Terms (MBAT) [9], and Geometric Analogue of Holographic Reduced Representations (GAHRR) [10]. Different representational schemes using high-dimensional vectors and operations on them are generally referred to as Vector Symbolic Architectures (VSA) [11] and the ultrahigh dimensionality is referred to as Hyperdimensional [3]. Table I summarizes different frameworks of VSAs.

B. General and Scalable Model of Computing

HD computing is a complete computational paradigm that is easily applied to learning problems. Its main difference from other paradigms is that it can operate with data represented as approximate patterns, allowing it to scale to large learning applications.

1) Applications of HD Computing: HD computing has been used commercially since 2008 for making semantic vectors for words – semantic vectors have the property that words with similar meaning are represented by similar vectors. The Random Indexing (RI) [13] algorithm for making semantic vectors was developed as an alternative to Latent Semantic Analysis (LSA) [14], which relies on compute-heavy Singular Value Decomposition (SVD). The original experiment used 37,000 "documents" on 7 topics to compute 8,000-dimensional semantic vectors of equal quality for 54,000 words.



(a) Text analytics (b) Electromyography (EMG)

Fig. 1. General and scalable HD computing for various cognitive tasks: (a) European languages recognition; (b) EMG-based hand gesture recognition.

SDV-based LSA requires memory in proportion to the product: 'size of vocabulary' × 'number of documents'. By contrast, RI requires memory in proportion to the size of the vocabulary, and the statistics of documents/contexts is learned through simple vector addition [13]. Thus, the complexity of the method grows linearly with the size of the training corpus and scales easily to millions of documents.

Multiplication and permutation make it possible to encode causal relations and grammar into these hypervectors, thereby capturing more and more of the meaning in language [4], [15]. We have also used HD computing successfully to identify the language of test sentences, as described at the start of this section [4], [16], and to categorize text [17]; other applications to text include common substrings search [18] and recognition of permuted words [19]. Using analog accelerometer signals, the feasibility for classifying vehicles was demonstrated in [20]. While these applications have a single input stream (Figure 1(a)), processing of biosignals from multiple sensors (Figure 1(b)) can also benefit from HD computing. For instance, we have adapted the architecture for text analytics to the classification of hand gestures, when analog electromyography (EMG) signals are recorded simultaneously by four sensors, as shown in Figure 1 (S_1–S_4) [21]. This architecture was further extended to operate on electroencephalography (EEG) data with 64 electrodes for a binary classification task [22].

2) Comparison With Conventional Machine Learning: Table II lists applications of HD computing to classification and recognition. It also compares the accuracy between HD computing and a baseline conventional machine learning methods as reported in the literature. For language identification, k Nearest Neighbor (k-NN) was considered as the baseline [16]. Text categorization was compared to various methods including Bayes, k-NN, and Support Vector Machine (SVM) in which SVM was the most accurate [23]. In EMG-based hand-gesture recognition, SVM was used as the standard of comparison [24]. For binary classification of EEG error-related potentials a Gaussian classifier was crafted by a skilled professional [22].

TABLE II

COMPARISON OF CLASSIFICATION ACCURACY BETWEEN HD COMPUTING AND CONVENTIONAL MACHINE LEARNING

Applications	Encoding	HD	Baseline
Language identification	n-gram	96.7%	k-NN, 97.9%
Text categorization	n-gram	94.2%	SVM, 86.4%
EMG gesture recognition	n-gram	97.8%	SVM, 89.7%
EEG error-related potentials	n-gram	74.5%	Gaussian, 69.5%
Activity recognition	sequence	82.2%	DT&ANN, 75.4%
Spoken-word classification	features	97.2%	HMM, 97.1%
Media-player prediction	sequence	36.0%	MOMM, 32.0%
Image classification	pixels	98.3%	HGN, 76.2%
Fault isolation	features	68.0%	k-NN, 71.0%
Vehicle classification	features	100.0%	none

HD computing has also been used for fusing data streams from multiple sources, including categorization of bodily physical activities [25], and prediction of mobile-phone use patterns [26]. In [25] the principles of HD computing were applied to human activity recognition using data from several heterogeneous sensors, and the results were compared to ones obtained by combining Decision Trees (DT) and Artificial Neural Network (ANN). Similar encoding scheme was used in [26] for predicting behavior of mobile-device users (e.g., media player prediction) and compared to results from a mixed-order Markov model (MOMM). In [27] HRR-like representations were formed directly from the features using random projection matrices; such representational scheme was found to be efficient for the task of spoken-word classification demonstrating performance comparable to Gaussian mixture-based continuous-density hidden Markov model (HMM).

In [28] Kleyko *et al.*proposed an encoding scheme using a Binary Spatter Code for forming distributed representations of arbitrary patters. In the test task of recognizing black-and-white images in the presence of noise, the scheme has shown improved performance compared to the Hierarchical Graph Neuron (HGN) which uses the same abstraction when representing patterns. The same scheme was also applied to the problem of data-driven fault isolation in an industrial plant in [29]. For language identification, HD computing was almost as accurate as the baseline, and for the other applications it surpasses the baseline methods. In every case the HD algorithm is as simple or simpler than the traditional algorithm.

Note that all aforementioned operations – multiplication, addition, permutation, distance measuring – are performed on hypervectors that can represent arbitrary patterns. The patterns can be approximate, and so the results will be approximate. The inherent robustness of high-dimensional distributed representation allows multiple alternatives to be superposed over the same vector and processed as a single unit. The ability to robustly compute with approximate patterns underlies the amenability and scalability of HD computing for a variety of cognitive applications.

C. Robustness of Computations

HD computing is extremely robust. Its tolerance for low-precision components (see Section IV-D) and faulty

components (see Section IV-E) is achieved by bio-inspired properties of hypervectors: (pseudo)randomness, high-dimensionality, and fully distributed holographic representations. Symbols represented with hypervectors begin with i.i.d. components and when combined with the Multiply-Add-Permute (MAP) operations, the resulting hypervectors also appear as identically distributed random vectors, and the independence of the individual components is mostly preserved. This means that a failure in a component of a hypervectors is not "contagious". At the same time, failures in a subset of components are compensated for by the holographic nature of the data representation i.e., the error-free components can still provide a useful representation that is similar enough to the original hypervector. This inherent robustness eliminates the need for asymmetric error protection in memory units. This makes HD data representation suited for operation at low signal-to-noise ratios (SNR).

D. Fast and Continuous Learning

In contrast to other neuro-inspired approaches in which learning is computationally much more demanding than subsequent classification, learning in HD computing is based on the same algorithms as classification. The HD algorithms are relatively lightweight and can be realized in low-energy devices. The algorithms work in "one-shot," namely, object categories are learned in a *single pass* over the training data. As an example, HD algorithm achieved a high level of classification accuracy (97.8%) in one pass over 1/3 the training data required by the state-of-the-art Support Vector Machine (SVM) on the same task [21].

Furthermore, the same representational scheme can be used for a variety of tasks with similar success rates. Because all are based on the same set of operations (add, multiply, permute, compare), it is possible to build a general-purpose computational engine for all these tasks. In addition, since the same algorithms are used for learning and classification, the architecture is ideal for continuous on-line learning.

E. Memory-Centric With Embarrassingly Parallel Operation

At its very core, HD computing is about manipulating and comparing large patterns within the memory itself. The MAP operations allow a high degree of parallelism by needing to communicate with only a local component or its immediate neighbors. Other operations such as the distance computation can be performed in a distributed fashion. This is a fundamental difference from traditional computational architectures, where data has to be transported to the processing unit and back, creating the infamous memory wall. In HD processing, logic is tightly integrated with the memory and all computations are fully distributed. This translates into substantial energy savings, as global interconnects are accessed at a relatively low frequency.

HD computing, therefore, has high application potential with novel memory-centric designs such as Intel–Micron 3D XPoint, Samsung In-Memory Database, and Altera Stratix 10 MX DRAM system-in-package that can be integrated at high density with programmable logic blocks. For example, the Stratix 10 MX DRAM system-in-package [30] meets the demanding memory bandwidth requirement by combining a high-performance monolithic FPGA fabric and high bandwidth memory modules, all in a single package.

III. 2D Architecture for HD Computing

As a concrete application of HD computing, let us look at an implementation of the language-recognition algorithm discussed in Section II, except that instead of bipolar vectors ($\{1, -1\}^D$) we will use the Binary Spatter Code [16]. The HD classifier generates trigram profiles as hypervectors and compares them for similarity. As shown in Figure 2, the design is based on a memory-centric architecture where logic is tightly integrated with the memory and all computations are fully distributed. The HD classifier has two main modules: encoding and similarity search. The encoding module projects an input text, composed of a stream of letters, to a hypervector in high-dimensional space. Then this hypervector is broadcast to the similarity-search module for comparison with a set of precomputed language hypervectors. Finally, the search module returns the language that has the closest match based on Hamming distance similarity.

A. Encoding Module

The encoding module accepts the text as a stream of letters and computes its representation as a hypervector. The module has an item memory that holds a random hypervector (the "letter" hypervector) for each of the 26 letters and the space. The item memory is implemented as a lookup table that remains constant. In the binary implementation of the encoding module, a letter hypervector has an approximately equal number of randomly placed 1s and 0s, and the 27 vectors are approximately orthogonal to each other.

The encoding module computes a hypervector for each block of 3 consecutive letters as the text streams in. It consists of 3 stages in FIFO style, each of which stores a letter hypervector. A trigram hypervector is created by successively permuting the letter vectors based on their order and binding them together, which creates a unique representation for each unique sequence of three letters. For example, the trigram "abc" is represented by the hypervector $\rho(\rho(A) \oplus B) \oplus C = \rho(\rho(A)) \oplus \rho(B) \oplus C$. Use of permutation and binding distinguishes the sequence "abc" from "acb", since a permuted hypervector is uncorrelated with all the other hypervectors.

The random permutation operation ρ is fixed and is implemented as a rotation to right by 1 position as shown in Figure 2. For instance, given the trigram "abc", the A hypervector is rotated twice ($\rho(\rho(A))$), the B hypervector is rotated once ($\rho(B)$), and there is no rotation for the C hypervector. Once "c" is reached, its corresponding C hypervector is fetched from the item memory and is written directly to the first stage of the encoder (i.e., Letter_3 hypervector in Figure 2). The two previous letters are rotated as they pass through the encoder and turn into $\rho\rho(A)$ and $\rho(B)$. Componentwise bindings are then applied between these three hypervectors to compute the trigram hypervector, i.e., $\rho\rho(A) \oplus \rho(B) \oplus C$.

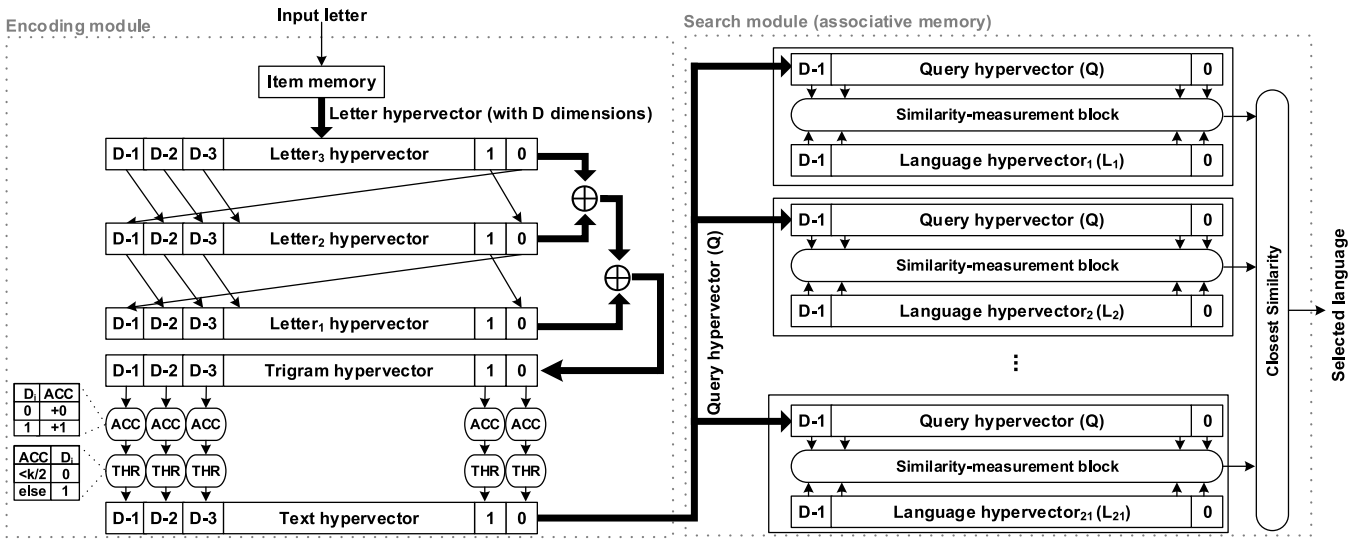


Fig. 2. HD classifier a 2D architecture for HD computing: encoding module and search module.

Since the trigram hypervector is binary, the binding between two hypervectors is implemented with D XOR gates.

The hypervector for the input text is computed by adding together the hypervectors for all the trigrams in the text and by applying a threshold. An input text of length $k+2$ generates k trigram vectors. We implement the componentwise addition with a set of D accumulators (ACC in Figure 2), one for each dimension of the hypervector, and count the number of 1s in that component location. This componentwise accumulation produces a D-dimensional vector of integers. To compute the corresponding binary vector, the encoding module applies a threshold of $k/2$ (implementing the majority function $(k, k/2)$) to every accumulator value, where k is the number of trigrams accumulated from the input. Left side of Figure 2 shows such a dedicated accumulation and thresholding for every hypervector component. The output of the encoding module is the binary text hypervector.

The encoding module is used for both training and testing. During training when the language of the input text in known, we refer to the text hypervector as a *language* hypervector. Such language hypervectors are stored in the search module. When the language of a text is unknown, as it is during testing, we call the text hypervector a *query* hypervector. The query hypervector is sent to the similarity search module to identify its source language.

B. Similarity-Search Module

The search module stores a set of language hypervectors that are precomputed by the encoding module. These language hypervectors are formed in exactly the same way as described above, by making the text hypervectors from samples of a known language. Therefore, during the training phase, we feed texts of a known language to the encoding module and save the resulting text hypervector as a language hypervector in the search module. We consider 21 European languages and at the end of training have 21 language hypervectors, each stored in its own row of the search module.

The language of an unknown text is determined by comparing its query hypervector to all the language hypervectors. This comparison is done in a distributed fashion using an associative memory, and with the Hamming distance as the similarity function.

Hamming distance counts the number of components at which two binary vectors disagree. The module uses a set of D XOR gates to identify mismatches between the two hypervectors. To ensure scalability, the similarity-measurement block compares only one component each clock cycle. Hence, it takes $O(D)$ cycles to compute the Hamming distance between the two hypervectors. This block is replicated 21 times (the number of languages in our application) within the search module as shown in Figure 2. The query hypervector is broadcast across the search module, hence all the similarity-measurement blocks compute their distance concurrently. Finally, a combinational comparison block selects the minimum Hamming distance and returns its associated language as the language that the unknown text has been written in.

IV. Experimental Results

In this section, we present our experimental results for the HD classifier as a 2D architecture. We first present our application of language recognition and its dataset. Next, we describe a conventional machine learning method as a baseline for comparison with the HD classifier. We provide RTL implementations for these two classifiers and compare their classification accuracy, memory footprints, energy consumption and robustness.

A. Language Recognition Dataset

We consider an application for recognition of 21 European languages, discussed in Section II. The sample texts are taken from the Wortschatz Corpora [31] where large numbers of sentences in these languages are available. We train each language hypervector based on about a million bytes of text. To test the ability of identifying the language of unseen text samples, we select test sentences from Europarl Parallel Corpus [32] as an independent text source. This corpus provides 1,000 samples

TABLE III
CLASSIFICATION ACCURACY AND MEMORY FOOTPRINT OF THE HD CLASSIFIER AND BASELINE CLASSIFIERS

	Accuracy		Memory (Kb)	
	HD	Baseline	HD	Baseline
Bigrams (n=2)	93.2%	90.9%	670	39
Trigrams (n=3)	96.7%	97.9%	680	532
Tetragrams (n=4)	97.1%	99.2%	690	13837
Pentagrams (n=5)	95.0%	99.8%	700	373092

of each language, and each sample is a single sentence, for a total of 21,000 sentences. The accuracy recognition metric used throughout this paper is the percentage of sentences identified correctly.

B. Baseline Machine Learning Method

As the baseline technique, we choose a nearest neighbor classifier that uses histograms of n-grams. To compute the distance between histograms, the dot product is used. A histogram is generated for each language to capture the frequency of n-grams in the training data. Hence, the outcome of the training phase is a set of 21 histograms that represent the languages. The histogram for each test sentence is generated in the same way. To find out the language of the test sentence, we compute the dot product of its histogram with the 21 precomputed histograms. The highest dot product score identifies the language that the test sentence is written in. Considering n-grams as the input features, a histogram requires L^n integer components where L is 27 in our application. To reduce the memory footprint, we convert the integer components of histograms to binary using their mean value as the threshold.

The nearest neighbor among histograms was chosen for two reasons. First, the histogram has full information about the n-gram statistics, so it sets the highest standard of comparison. Second, from a hardware point of view, nearest neighbor is equally suited for the HD algorithm. Both use n-grams of letters, and the operations have equal complexity. For instance, computing the frequency of an n-gram in the baseline is a lookup action followed by addition. For finding the best match, both use dot product to measure distance, and use the Hamming distance when the histograms are reduced to binary. Essentially, this baseline uses the same hardware components as the HD architecture but excludes the item memory. In the following sections we compare the methods in detail.

C. Classification Accuracy and Memory Usage

Table III compares the two classifiers when the histograms have been reduced to binary. The first two columns show the classification accuracy with different n-grams and the last two columns show the memory footprint. With histograms based on bigrams, the baseline has 2.3% lower recognition accuracy than the HD classifier. However, using n-grams with $n \geq 3$, the baseline is slightly more accurate. For example, the baseline shows 97.9% recognition while the HD classifier shows 96.7% for trigrams. In this case, the HD classifier requires 1.2× as many memory cells as the baseline. On the upside, the HD classifier is able to represent many more n-grams within the same hardware structure. It scales very well: for instance, by moving from trigrams ($n = 3$) to pentagrams ($n = 5$), the HD classifier must add memory cells for only 2 extra hypervectors (in the encoding module) whereas the memory required by the baseline grows exponentially with n. Using pentagrams of letters, the baseline shows an accuracy of 99.8% (4.8% higher than the HD classifier), at the expense of 500× larger memory size. Of course the exact counts of all pentagrams that appear in a million bytes of text can be captured in much less memory than that but the algorithm is no longer as simple.

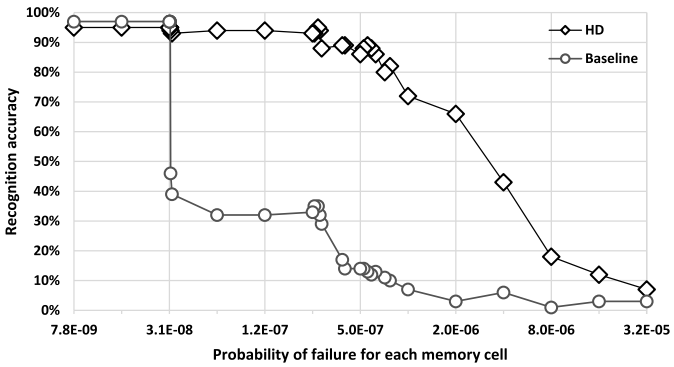


Fig. 3. Accuracy of HD classifier and baseline with faulty memory cells.

D. Robustness in the Presence of Low-Precision Components

Here we assess the robustness of HD classifier in language recognition by replacing high-precision components with low-precision ones, down to binary, in the search module (see Section III-A and Figure 2).

Each component of the text hypervector requires a multibit cell memory before thresholding. For learning a megabyte of text, each hypervector component will need 19 bits precision for summing up a million of random 0s and 1s. We observe that the recognition accuracy is slightly decreased by reducing the bitwidth (i.e., the precision of each component). Such a robust behavior enables us to turn the high-precision hypervectors to binary hypervectors of the same dimensionality, while slightly degrading the recognition accuracy from 97.4% to 96.7%. Reducing bitwidth reduces the amount of memory required by the search module by a factor of 19×. Moreover, comparing binary hypervectors requires fewer hardware resources in the search module.

E. Robustness Against Memory Errors

Here we assess the classifiers' tolerance for memory errors. We target RTL fault simulations where we inject memory bit flips during every clock cycle of execution. We consider a wide range of probability of failures for each memory cell; the fault simulations cover all the memory elements in both designs.

Figure 3 shows recognition accuracy with erroneous memory cells; the X-axis displays the probability of failure for each memory cell in every clock cycle. The baseline is able to maintain its high accuracy of 97% using faulty memory

cells with the probability of failure at 3.16E-08 and lower values. At 3.17E-08 the accuracy falls sharply to below 46%. However, the HD classifier is very robust: it maintains recognition accuracy of 94% and higher for probability of failure up to 2.78E-07. At or near peek performance (94% for the HD classifier and 97% for the baseline), the HD classifier tolerates 8.8-fold probability of failure compared to the baseline. By further increasing the probability of failure by 2.8×, to 7.69E-07, the HD classifier is still 80% accurate or better. Finally, the accuracy of the HD classifier drops to 43% when the memory cells fail with probability 4.00E-06, i.e., ≈120× higher than the failure rate that the baseline could tolerate for the same accuracy.

We further analyzed the effect of bit-flipping on the decoding of HD vectors. The task consists of storing a sequence of letters in a single hypervector and retrieving a letter from a specified position in the sequence [33]. A sequence of M letters can be stored in a hypervector by encoding each letter's order with permutation and by superposing (adding together) the resulting letter vectors into a single trace vector, H. The elements of the superposition hypervector are made binary by the majority rule and hence the trace hypervector H is of the form

$$H = [\sum_{\mu=1}^{M} \rho^{\mu}(X_{\ell_\mu})] \tag{1}$$

Due to the properties of addition, the trace hypervector is similar to each of the added vectors. Hence, each element of the letter sequence can be decoded from the trace hypervector by first inverting the permutation that encodes the letter's position position m as $\rho^{-m}(H)$ and then finding the most similar hypervector amongst the letters stored in the item memory. The similarity between two hypervectors is measured by Hamming distance normalized by D, Δ_H. The majority function preserves only partial similarity to its constituent hypervectors. In other words, Hamming distance between the "unpermuted" hypervector $\rho^{-m}(H)$ and the original hypervector X_{ℓ_m} in position m increases with the length of the letter sequence. For a sequence of length M, the mean value of Hamming distance Δ_{H_M} between $\rho^{-m}(H)$ and X_{ℓ_m} is given in [34] as

$$\Delta_{H_M} = \frac{1}{2} - \binom{M-1}{\frac{1}{2}(M-1)} / 2^M \tag{2}$$

The standard deviation around the mean depends on the dimensionality, D, and is calculated as

$$\sigma = \sqrt{0.25D} \tag{3}$$

We characterize the noise in the trace hypervector by probability p_f of flipping a single bit. Noise in the trace hypervector is evenly distributed among the components and it drives Hamming distances toward 0.5. The larger the number of vectors in the trace vector, the further it is from the individual vectors, and the less are Hamming distances affected by bit-flipping. In fact, random flipping has no effect on the mean distance between vectors that are 0.5 apart. The growth of

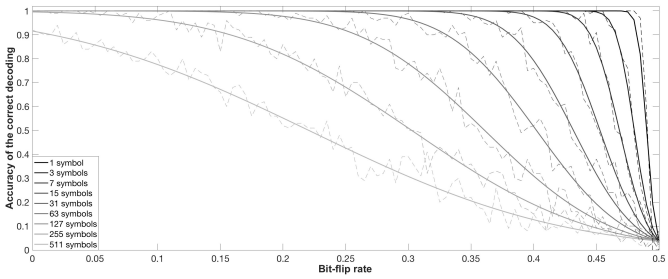


Fig. 4. Probability of correctly decoding a letter, p_{corr}, from the trace hypervector, as a function of bit-flip rate p_f for different sequence lengths $M = 2^k - 1$, $k \in [1, 9]$; $D = 10000$; $L = 27$. Solid lines are analytical values according to (6). Dashed lines are results of simulations averaged over 100 runs.

Hamming distance between $\rho^{-m}(H)$ and X_{ℓ_m}, when the trace hypervector contains Dp_f flipped bits, is given by

$$\Delta_{H_f} = (1 - 2\Delta_{H_M})p_f \tag{4}$$

Thus, the total Hamming distance is:

$$\Delta_{H_t} = \Delta_{H_M} + \Delta_{H_f} \tag{5}$$

If Δ_{H_t} is close to 0.5, the probability of incorrectly decoding the letter in a specified position is high. The probability of correct decoding p_{corr} was derived in [33] and is given by

$$p_{corr} = \int_{-\infty}^{\infty} \frac{dh}{\sqrt{2\pi}\sigma} e^{-\frac{(h-(0.5-\Delta_{H_t})D)^2}{2\sigma^2}} \left[\Phi\left(\frac{h}{\sigma}\right)\right]^{L-1} \tag{6}$$

where L is the number of letters in the alphabet, and Φ is the cumulative distribution function of the normal distribution. The accuracy of decoding a single letter from the trace hypervector, p_{corr}, against the bit-flip rate p_f for several sequence lengths is shown in Figure 4. Note that the curves derived analytically from (6) match the simulations. Also note that as the sequence length M increases and Δ_{H_M} approaches 0.5, the effect of flipping bits becomes more gradual. However, even long sequences of hypervectors (63 symbols) demonstrate superb accuracy for bit-flip rates up to 0.15. When p_f equals 0.5, the trace hypervector becomes unrelated to any of its constituent vectors, and the accuracy of decoding equals a random guess, i.e., $p_{corr} = \frac{1}{L}$ for any sequence length M.

F. Energy Efficiency

We use a standard ASIC flow to design dedicated hardware for the baseline classifier and the HD classifier. We describe the classifiers in a fully parameterized manner, using RTL SystemVerilog. We apply identical constraints and flow to both designs. For the synthesis, we use *Synopsys Design Compiler* with TSMC's 65 nm LP CMOS process. We extract the switching activity of these classifiers during postsynthesis simulations in *ModelSim* using the test sentences. Finally, we measure their power consumption using *Synopsys PrimeTime* at (1.2V, 25 °C, TT) corner.

Figure 5 shows the power consumption and area for various dimensionality of the three main components of the HD classifier: the item memory, the encoder, and the associative memory. The area results indicate that the size of the HD

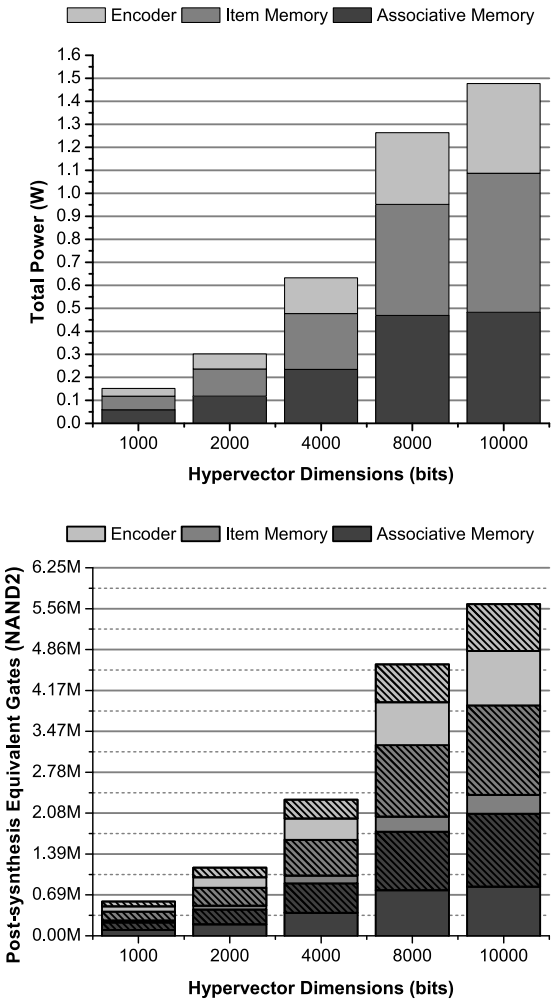


Fig. 5. Breakdown of area and power consumption of the HD classifier for computing trigram profiles with hypervectors of various dimensionality. The hatched parts refer the area for non-combinational logic.

classifier and its components increase almost linearly with the size of the hypervectors. This is in contrast to conventional methods where the size of the dominant components such as multipliers and other complex operators increases polynomially with the width of the datapaths. Therefore, a HD classifier is expected to be much smaller and consume far less energy than comparable conventional methods.

As shown in the Figure, the item and associative memories together constitute more than two-thirds of the size and power consumption; only the encoder has a substantial portion of its resources devoted to combinational logic. This highlights the fact that the HD classifier is largely a memory-centric machine. Hence, current trends in memory technology are a natural fit for HD computing: the hypervectors can be efficiently stored in dense crossbars with in-memory logic that can perform the encoder operations. The simulation of average switching activity per net reveals that the encoder is almost three times as active as the rest of the design. Therefore, a further improvement in power consumption can be achieved by pursuing sparse hypervectors and operations that preserve sparsity.

We compare the power consumption for trigrams only, since with n-grams of $n \geq 4$, the baseline classifier becomes increasingly less efficient compared to the HD classifier due to the exponential growth in the amount of memory required. With 47% of the energy required by the baseline classifier, the HD classifier is only 1.2% less accurate: 96.7% versus 97.9% for the baseline. Although both designs use binary components and low-cost Hamming distance for similarity measurement, the HD classifier achieves higher energy efficiency thanks to its one-shot computation with highly scalable and local operations. We should note that this energy saving is achieved without harnessing the robustness of HD computing demonstrated in Section IV-E.

V. 3D Architecture for HD Computing

In this section we explore how to translate the properties of HD computing into a functional and efficient 3D memory-centric realization, and highlight the challenges. We focus on non-volatile memory technologies that can be integrated at high density and require little energy. Such an efficient realization constrains the underlying technology in several ways:

A. Tight Integration of Memory and Logic

The proposed HD computing architecture merges computation and storage into a single fabric. While this can be accomplished in a traditional 2D process, the increase in cell size would lead to substantial energy overhead. A 3D approach where logic and memory are stacked on top of each other leads to a far more efficient realization [35], [36]. In fact, multiple layers of such approach are envisioned.

B. Non-Volatile, Multi-Level Memory

Because the realization is memory-dominated and memory accesses are sparse in time and space, it is essential that memory cells (and logic) be powered down when not in use. Otherwise leakage power will dominate the overall power consumption of the system. This means that the associative memory should be implemented in non-volatile technology. Observe that memory reads (classification) are far more frequent than memory writes (learning), which is consistent with the energy requirements of these operations in most non-volatile memory technologies. In addition, data values do not need to be binary. They could be multi-level [37] continuous or discrete, depending on the HD framework, the latter being consistent with multi-layer cell technology.

C. Low-Voltage Operation

A dominant part of the energy dissipated in writing to and reading from the memory is due to the CV^2 loss in the interconnects. Therefore, the ability to operate the memory at low voltages while tolerating the resulting errors is of paramount importance.

Realizing the full potential of HD computing requires a generic architecture that enables the following: (1) multiple

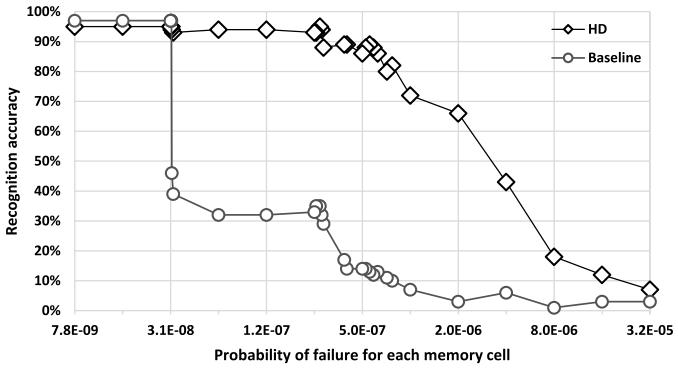


Fig. 6. Block diagram of HD computing generic architecture.

applications on the same hardware; (2) learning and classification/execution using the same algorithm; (3) availability of all operations that define HD computing; and (4) achievement of high energy efficiency by exploiting robust algorithms that tolerate errors from operating at very low SNRs (<10), on-demand computation with zero-leakage at stand-by, in-memory processing, and non-volatile state storage, e.g., [38]–[41]. Ease of programming is another essential component of the HD architecture, being based on an abstract model of computing supported by a rich algebra, which in principle can lead to simple mapping of function to implementation assuming that the underlying architecture matches the computational model.

Figure 6 shows an overview of a generic HD computing architecture with different modules. Note the parallel configuration of the encoder and associative memory modules. This allows an overlay of applications and integration in learning. The front-end feature extractors are strongly dependent on the "signals to be learned", i.e., the application. The inner modules work on the initially extracted (crude) features in a generic manner by exploiting the HD computing theory, making the hardware application-agnostic. We note, however, that the multi-layer associative memory can be highly beneficial for the feature extractors by themselves [42].

1) Architectural Design: As can be seen in Figure 6, the generic architecture consists of three kinds of modules:

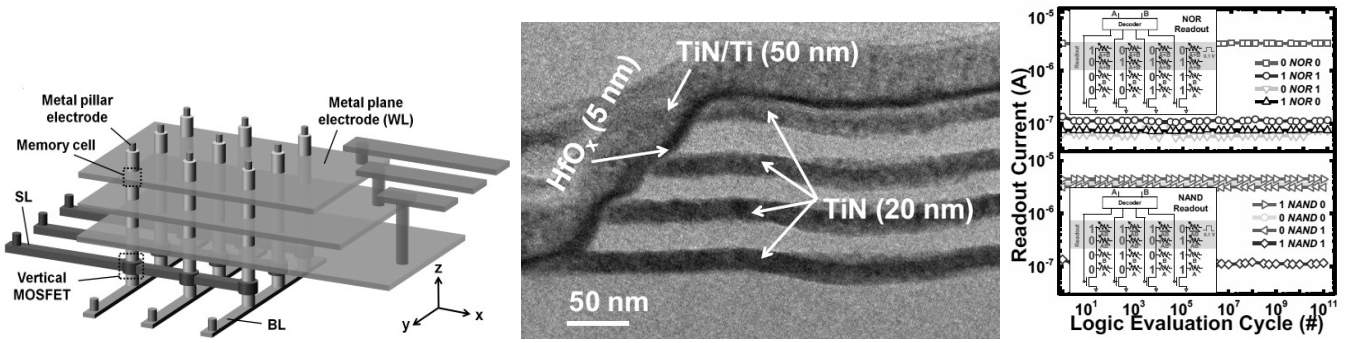
a) HD encoders/decoders: These modules map the input data onto high-dimensional vectors, perform operations on those hypervectors to encode sequences, associations, and patterns, and extract features by performing reverse operations for a subsequent layer in a deep net. Operations are performed in a distributed and parallel fashion. The most general approach is to realize those functions in a memory-rich fabric where the data streams through layers of operations with the function of each layer being determined by the local (non-volatile) memory. To obtain energy efficiency in this massively parallel architecture, the following requirements must be met: (1) low SNR operation using either ultra-low-voltage or current-mode operation; (2) sparse data representations so that only few cells at each layer are activated per cycle; (3) on-demand operation with zero stand-by power.

b) Associative memories: These modules store high-dimensional vectors and perform the matching of incoming vectors to stored vectors. Operations in the associative memory fall under a number of different strategies: simple storage of patterns; incremental adjustment of memory contents based on the applied input patterns; simple and distributed match; auto-association (for noise removal); and more. Data in the memory can be binary or discrete (with an accuracy level of 4 bits mostly sufficient). Efficient realization of the memory requires (1) non-volatility to support low stand-by power; (2) multi-level memory enabled by stacking memory cells similar to the advanced NAND Flash memories but operating at much lower voltages – such as potentially offered by RRAM; and (3) distributed and tightly integrated match computations.

c) Inter-module interconnect: The HD computing architecture shown in Figure 6 consists of an array of encoder/decoder blocks and memory functions. This partitioning into multiple sub-modules is necessary for a number of reasons: (1) implementation efficiency – too large modules come with energy overhead and degradation from noise; hence partitioning of the memories along the row and column lines is crucial; (2) hierarchy – typical HD application will operate on multiple layers of data abstraction (just like most cognitive systems). Each of these layers operates with its own data encoding and learned patterns; (3) diversity – a simple computational engine may combine various types of input data and data models (just like the brain does). While it is perfectly reasonable to multiplex those on the same fabric, most often it is more efficient to distribute the operations in space and combine the results at the higher abstraction layers. How to provide an efficient and adaptable interconnect structure between the modules is a major challenge. The associative memories used as programmable interconnect matrices may be a viable strategy.

2) Resistive Memory Devices: Resistive random access memory presents a compelling opportunity as it uses materials compatible with Si CMOS and fabrication temperatures below 400°C. Such resistive devices can be based on oxygen vacancies (RRAM) [45] that form conductive filaments or metal ions, creating conductive bridges through a solid electrolyte or an oxide (CBRAM). Importantly, RRAM can be integrated in 3D. For example, in the implementation shown in Figure 7(a) and practically demonstrated in Figure 7(b) [43], the 3D RRAM performed on par with its planar 2D crosspoint counterparts. Recent advances [46], [47] point to the possibility that a 128-layer stacked 3D memory will provide 64Tb on-chip memory for a 5 nm half-pitch technology.

a) Resistive in-memory computing: In-memory computing operations are natural to the 3D architecture. It has been shown that any arbitrary multi-stage Boolean expression can be implemented by programming 3D RRAM cells along a common pillar [44]. Owing to the unique-common pillar structure, the logic operations are readily realized on a multi-layer 3D RRAM, where the state variable for Boolean logic is the RRAM resistance values. Two computing modes are available: programming mode and read-out mode. Specific programming pulse trains are used for basic Boolean operations, such NAND, NOR, and bit shift. Notably, read-out mode is employed for frequently used logic where 10^{11}-cycle operations for NAND/NOR logic evaluations were measured



(a) (b) (c)

Fig. 7. Four-layer 3D vertical RRAM with in-memory computing. (a) Schematic of a 3D RRAM array [43]. (b) Image of four-layer 3D vertical RRAM [43]. (c) In-memory computing of memorized functions [44].

experimentally, limited merely by test time (Figure 7(c)). Additionally, the in-memory computing on 3D RRAM is dynamically reconfigurable.

The attributes described above suggest that RRAM technology is ideal for implementing HD computing. We can build multi-layer RRAM stacks where the HD computing operations are carried out by 3D RRAM cells in memory, and where local access is provided to multibit storage cells. It may be possible to reduce operating voltages further by exploiting the fact that HD computing is possible at low SNRs. In addition, the retention time can be smaller than that required for a storage class memory (ca. 10 years). Therefore, a small ON/OFF ratio could be good enough. Since a conductive-bridge memory (CBRAM) has proven to be programmable at very low voltages, its shorter retention time could be sufficient for working memory (e.g., in the encoding module) of a HD computer.

VI. Algorithm Development and Applications

Possible applications of high-dimensional computing go beyond the simple examples discussed above. Those examples deal with the encoding of frequencies and probabilities of occurrence into hypervectors, and with the comparison of the resulting vectors. However, much of ordinary computing is concerned with the *structure* of information and with meaning conveyed by structure. The importance of structure is obvious in language. For example, the statement "John is here" becomes a question by changing the order of the first two words. HD computing is fully able to deal with structure, thanks to the MAP operations and their underlying algebra. Language and visual perception offer ample opportunity to develop algorithms for dealing with structure.

A. Language Understanding

Traditional methods of computational linguistics are of two basic kind, statistical and structural. Latent Semantic Analysis (LSA) [14] is an example of the statistical; it computes semantic vectors for words from word co-occurrence statistics in large text corpora. Words with similar meaning end up with similar semantic vectors, but the vectors do not distinguish between grammatical types. The semantic vectors for "hot" and "fire" made with LSA are similar but contain no information about one being an adjective and the other a noun, or a verb in some contexts. The structural approach, in turn, is based on a grammar and stumbles on multiple ways to parse a given string of words. Where a human mind sees only one way to assemble the words into a meaningful sentence, a computer cannot decide without a good way to represent meaning.

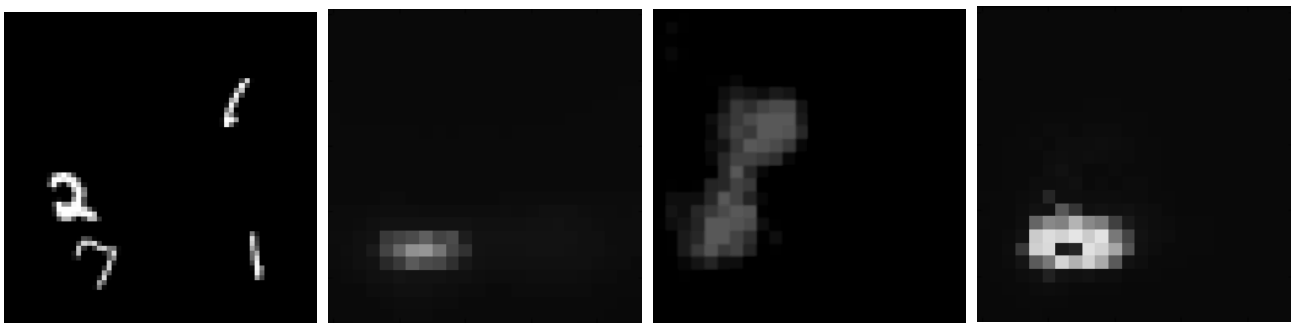
We expect to get much closer to human performance by computing with hypervectors. The vectors can encode similarity of meaning in the manner of LSA and embed it in a structured representation by binding with multiplication. Multiple alternatives can be encoded into a single vector and the ambiguities resolved based on subsequent information – again in the form of a vector that supports one alternative over the others. The operations on hypervectors allow it in a manner that is transparent and mathematically rigorous.

B. Visual Scene Understanding

Language in text form provides an easy entry into HD computing because the data consist of discrete symbols (letters and words) that are readily mapped to high-dimensional vectors. Other kinds of data such as images and sound waveforms call for a hybrid approach where HD computing is combined with deep learning. Deep learning (either supervised or unsupervised) is used to discover features in the data that allow its mapping into high-dimensional vectors. These vectors are then combined and manipulated within the HD framework for high-level reasoning tasks.

We will illustrate the idea with visual scene understanding – i.e., parsing an image into its constituent parts and representing relations among them. A simple example of such a task is shown in Figure 8. Figure 8(a) shows several digits in different locations within an image. The task is to take the image as input and to form a representation of it that allows questions such as "what is below a 2 and to the left of a 1?" to be answered. This is a challenging problem for traditional neural networks because it involves reasoning about the relations between objects rather than simply learning associations between images and labels.

HD computing solves it by representing the scene as a collection of bindings between "what" and "where." A window



(a) (b) (c) (d)

Fig. 8. Understanding a visual scene "what is below 2 and to the left of 1?". (a) Example image (b) "Below a 2". (c) "To the left of a 1". (d) Combined.

of attention is focused on different locations in the scene, and at each location the contents ("what") are fed through a deep network to form a hypervector for the visual pattern in the window. The location is also encoded as a hypervector. Both vectors are learned so that when they are bound together, and then superposed with the other "what" and "where" bindings from the other locations in the image, they result is a *scene vector* for the image. The scene vector can then be queried for objects and their locations. A query is also represented as a hypervector, and the answer is found by multiplication and clean-up. For example, "below a 2" is represented as the high-dimensional embedding of "2" bound with a vector representing "below". Multiplied by the scene vector then yields the activation map shown in Figure 8(b). When that is combined with the activation map for "to the left of a 1" (see Figure 8(c)), the result (see Figure 8(d)) specifies the location of the digit seven with high confidence. One further multiplication (by the scene vector) identifies it as seven.

VII. Conclusion

Computing with high-dimensional vectors has long been a part of cognitive science and is at the core of artificial neural nets, with linear algebra as the underlying math. The present paper goes a step further. It is based on the notion that the operations on the vectors form an algebraic structure resembling a field, and that computing be understood in terms of the algebra – it becomes a lot like ordinary arithmetic. That idea was developed fully in Plate's Holographic Reduced Representation in the early 1990s [12]. It has been shown since then that computing of this kind can be supported by high-D vectors of different kinds – that high dimensionality is more important than the nature of the dimensions. In this paper we have mostly used binary vectors.

The conventional (von Neumann) model of computing is deterministic, and the engineering and manufacturing effort to make computer circuits reliable is considerable. It is also costly in material and energy. By contrast, HD computing uses randomness constructively and tolerates variation and errors in many of the components. Several experiments in this paper make that point.

HD algorithms are often simpler and scale better than conventional algorithms for the same task. We see that with semantic vectors and language identification, and in general in machine learning from large volumes of streaming data. Furthermore, the HD algorithms are ideal for parallel implementation, making high throughput possible with systems built of slow components.

Looking into the future of computing, it is a curious fact that the nanomaterials and structures studied in technology laboratories worldwide, have many properties that match the needs of HD computing. We will have circuits for 10,000-bit words if our algorithms need them. But how ever successful HD computing will be, it will not replace conventional computing, merely extend the range of tasks possible for computes. Obvious candidates include language understanding and image understanding.

Acknowledgment

The authors would like to thank H.-S. Philip Wong, Sayeef Salahuddin, and Eric Weiss.

References

[1] S. Williams and E. P. DeBenedictis, "OSTP nanotechnology-inspired grand challenge: Sensible machines (extended version 2.5)," Tech. Rep., Oct. 2015.

[2] D. Rossi *et al.*, "PULP: A parallel ultra low power platform for next generation IoT applications," in *Proc. IEEE Hot Chips 27 Symp. (HCS)*, Aug. 2015, pp. 1–39.

[3] 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, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1007/s12559-009-9009-8

[4] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *Proc. 10th Int. Conf., Quant. Interact.*, San Francisco, CA, USA, Jul. 2016, pp. 265–274. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-52289-0_21

[5] T. A. Plate, *Holographic Reduced Representation*. Stanford, CA, USA: CLSI Publications, 2003.

[6] P. Kanerva, "Binary spatter-coding of ordered K-tuples," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 1996, pp. 869–873.

[7] R. W. Gayler, "Multiplicative binding, representation operators & analogy," in *Advances in Analogy Research: Integration of Theory and Data From the Cognitive, Computational and Neural Sciences*, D. Gentner, K. J. Holyoak, and B. N. Kokinov, Eds. Sofia, Bulgaria: New Bulgarian Univ., 1998, pp. 1–4. [Online]. Available: http://cogprints.org/502/

[8] D. A. Rachkovskij, "Representation and processing of structures with binary sparse distributed codes," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 2, pp. 261–276, Mar./Apr. 2001.

[9] S. I. Gallant and T. W. Okaywe, "Representing objects, relations, and sequences," *Neural Comput.*, vol. 25, no. 8, pp. 2038–2078, 2013.

[10] D. Aerts, M. Czachor, and B. De Moor, "Geometric analogue of holographic reduced representation," *J. Math. Psychol.*, vol. 53, no. 5, pp. 389–398, 2009.

[11] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," in *Proc. Joint Int. Conf. Cognit. Sci. (ICCS/ASCS)*, 2003, pp. 133–138.

[12] T. A. Plate, "Holographic reduced representations," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 623–641, May 1995.

[13] P. Kanerva, J. Kristoferson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proc. 22nd Annu. Conf. Cognit. Sci. Soc.*, 2000, p. 1036. [Online]. Available: http://www.rni.org/kanerva/cogsci2k-poster.txt

[14] T. K. Landauer and S. T. Dutnais, "A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge," *Psychol. Rev.*, vol. 104, no. 2, pp. 211–240, 1997.

[15] G. Recchia, M. Sahlgren, P. Kanerva, and M. N. Jones, "Encoding sequential information in semantic space models: Comparing holographic reduced representation and random permutation," *Comput. Intell. Neurosci.*, vol. 2015, pp. 1–18, 2015.

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

[17] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2016, p. 1. [Online]. Available: https://www.date-conference.com/system/files/file/date16/ubooth/37923.pdf

[18] D. Kleyko and E. Osipov, "On bidirectional transitions between localist and distributed representations: The case of common substrings search using vector symbolic architecture," *Procedia Comput. Sci.*, vol. 41, pp. 104–113, Dec. 2014.

[19] D. Kleyko, E. Osipov, and R. W. Gayler, "Recognizing permuted words with vector symbolic architectures: A Cambridge test for machines," *Procedia Comput. Sci.*, vol. 88, pp. 169–175, Dec. 2016.

[20] D. Kleyko and E. Osipov, "Brain-like classifier of temporal patterns," in *Proc. 2nd Int. Conf. Comput. Inf. Sci. (ICCOINS)*, 2014, pp. 1–6.

[21] A. Rahimi, S. Benatti, P. Kanerva, 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, pp. 1–8.

[22] A. Rahimi, P. Kanerva, J. R. del Millán, and J. M. Rabaey, "Hyperdimensional computing for noninvasive brain–computer interfaces: Blind and one-shot classification of EEG error-related potentials," in *Proc. 10th ACM/EAI Int. Conf. Bio-Inspired Inf. Commun. Technol. (BICT)*, 2017, pp. 19–26.

[23] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *Proc. 10th Eur. Conf. Mach. Learn. (ECML)*, London, U.K., 1998, pp. 137–142. [Online]. Available: http://dl.acm.org/citation.cfm?id=645326.649721

[24] S. Benatti *et al.*, "A versatile embedded platform for EMG acquisition and gesture recognition," *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 5, pp. 620–630, Oct. 2015.

[25] O. Rasanen and S. Kakouros, "Modeling dependencies in multiple parallel data streams with hyperdimensional computing," *IEEE Signal Process. Lett.*, vol. 21, no. 7, pp. 899–903, Jul. 2014.

[26] O. J. Räsänen and J. P. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 9, pp. 1878–1889, Sep. 2016.

[27] O. Rasanen, "Generating hyperdimensional distributed representations from continuous valued multivariate sensory input," in *Proc. 37th Annu. Meeting Cognit. Sci. Soc.*, 2015, pp. 1943–1948.

[28] D. Kleyko, E. Osipov, A. Senior, A. I. Khan, and Y. A. Şekerciogğlu, "Holographic graph neuron: A bioinspired architecture for pattern processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 6, pp. 1250–1262, Jun. 2017.

[29] D. Kleyko, E. Osipov, N. Papakonstantinou, V. Vyatkin, and A. Mousavi, "Fault detection in the hyperspace: Towards intelligent automation systems," in *Proc. IEEE Int. Conf. Ind. Inf. (INDIN)*, Jul. 2015, pp. 1–6.

[30] M. Deo, J. Schulz, and L. Brown, "Stratix 10 MX devices solve the memory bandwidth challenge," Altera, San Jose, CA, USA, White Paper 01264-1.0, May 2016.

[31] U. Quasthoff, M. Richter, and C. Biemann, "Biemann C., 'Corpus portal for search in monolingual corpora,"' in *Proc. 5th Int. Conf. Lang. Resour. Eval.*, 2006, p. 21.

[32] P. Koehn. (2005). *Europarl: A Parallel Corpus for Statistical Machine Translation*. [Online]. Available: http://www.statmt.org/europarl/

[33] E. P. Frady, D. Kleyko, P. Kanerva, and F. T. Sommer, "The capacity of active memory the information capacity of distributed neural activity," *Redwood Center Preprint*, to be published.

[34] P. Kanerva, "Fully distributed representation," in *Proc. Real World Comput. Symp. (RWC)*, 1997, pp. 358–365.

[35] H. Li *et al.*, "Hyperdimensional computing with 3D VRRAM in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *IEDM Tech. Dig.*, Dec. 2016, pp. 16.1.1–16.1.4.

[36] F. Zhou, L. Guckert, Y.-F. Chang, E. E. Swartzlander, Jr., and J. C. Lee, "Bidirectional voltage biased implication operations using SiO_x based unipolar memristors," *Appl. Phys. Lett.*, vol. 107, no. 18, p. 183501, 2015.

[37] F. Zhou, Y.-F. Chang, B. Fowler, K. Byun, and J. C. Lee, "Stabilization of multiple resistance levels by current-sweep in SiO_x-based resistive switching memory," *Appl. Phys. Lett.*, vol. 106, no. 6, p. 063508, 2015.

[38] L. Ji *et al.*, "Integrated one diode–one resistor architecture in nanopillar SiO_x resistive switching memory by nanosphere lithography," *Nano Lett.*, vol. 14, no. 2, pp. 813–818, 2014.

[39] T.-J. Chu *et al.*, "Charge quantity influence on resistance switching characteristic during forming process," *IEEE Electron Device Lett.*, vol. 34, no. 4, pp. 502–504, Apr. 2013.

[40] C.-C. Hsieh, A. Roy, A. Rai, Y.-F. Chang, and S. K. Banerjee, "Characteristics and mechanism study of cerium oxide based random access memories," *Appl. Phys. Lett.*, vol. 106, no. 17, p. 173108, 2015.

[41] Y.-F. Chang *et al.*, "Understanding the resistive switching characteristics and mechanism in active SiO_x-based resistive switching memory," *J. Appl. Phys.*, vol. 112, no. 12, p. 123702, 2012.

[42] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Aug. 2013.

[43] H.-Y. Chen, S. Yu, B. Gao, P. Huang, J. Kang, and H.-S. P. Wong, "HfOx based vertical resistive random access memory for cost-effective 3D cross-point architecture without cell selector," in *IEDM Tech. Dig.*, Dec. 2012, pp. 20.7.1–20.7.4.

[44] H. Li and *et al.*, "Four-layer 3D vertical RRAM integrated with FinFET as a versatile computing unit for brain-inspired cognitive information processing," in *Proc. IEEE Symp. VLSI Technol.*, Jun. 2016, pp. 1–2.

[45] H. S. P. Wong *et al.*, "Metal oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

[46] B. Govoreanu *et al.*, "Vacancy-modulated conductive oxide resistive RAM (VMCO-RRAM): An area-scalable switching current, self-compliant, highly nonlinear and wide on/off-window resistive switching cell," in *IEDM Tech. Dig.*, Dec. 2013, pp. 10.2.1–10.2.4.

[47] S. Lee, J. Sohn, H.-Y. Chen, and H.-S. P. Wong, "Metal oxide-resistive memory using graphene edge-electrodes," *Nature Commun.*, vol. 6, Sep. 2015, Art. no. 8407.

Abbas Rahimi received the B.S. degree in computer engineering from the University of Tehran, Tehran, Iran, in 2010, and the M.S. and Ph.D. degrees in computer science and engineering from the University of California at San Diego, La Jolla, CA, USA, in 2015. He is currently a Post-Doctoral Scholar with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, USA. He is a member of the Berkeley Wireless Research Center and collaborating with the UC Berkeley's Redwood Center for Theoretical Neuroscience.

His research interests include brain-inspired computing, approximate computing, massively parallel integrated architectures, and embedded systems and software with an emphasis on improving energy efficiency and robustness. His doctoral dissertation has been selected to receive the 2015 Outstanding Dissertation Award in the area of New Directions in Embedded System Design and Embedded Software from the European Design and Automation Association. He has also received the Best Paper at BICT, in 2017, and the Best Paper Candidate at DAC, in 2013.

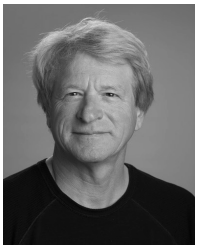
Sohum Datta received the bachelor's degree in electrical engineering from IIT Kanpur, Kanpur, India, in 2015. He is currently pursuing the Ph.D. degree in electrical engineering and computer sciences with the University of California at Berkeley, advised by Prof. J. M. Rabaey. His main interests are brain-inspired memory and cognitive models, stochastic computing, and energy-efficient systems implementing such algorithms.

Denis Kleyko received the bachelor's degree (Hons.) in telecommunication systems and the master's degree (Hons.) in information systems from the Siberian State University of Telecommunications and Information Sciences, Novosibirsk, Russia, in 2011 and 2013, respectively. He is currently pursuing the Ph.D. degree with the Dependable Communication and Computation Systems Group, Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden. His current research interests include vector symbolic architectures, high-dimensional computing, bio-inspired cognitive architectures, and machine learning.

Edward Paxon Frady received the B.S. degree in computation and neural systems from Caltech in 2008 and the Ph.D. degree in neuroscience from UC San Diego in 2014. He currently holds a postdoctoral position with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, where he is involving in hyperdimensional computing, machine-learning, and large-scale calcium imaging data analysis.

Bruno Olshausen received the B.S. and M.S. degrees in electrical engineering from Stanford University and the Ph.D. degree in computation and neural systems from the California Institute of Technology. From 1996 to 2005, he was an Assistant Professor and subsequently an Associate Professor with the Departments of Psychology and Neurobiology, Physiology and Behavior, UC Davis. Since 2005, he has been with the University of California at Berkeley (UC Berkeley), Berkeley. He is currently a Professor of Neuroscience and Optometry with UC Berkeley. He also serves as the Director of the Redwood Center for Theoretical Neuroscience, an interdisciplinary research group focusing on mathematical and computational models of brain function. His research aims to understand the information processing strategies employed by the brain for doing tasks, such as object recognition and scene analysis. The aim of this work is not only to advance our understanding of the brain, but also to discover new algorithms for scene analysis based on how brains work.

Pentti Kanerva received the Ph.D. degree in philosophy from Stanford University. He was involved in designing and building computer systems for 20 years, and he has over 30 years of research into understanding brains in computing terms. He has held research positions at the NASA Ames Research Center, the Swedish Institute of Computer Science, and the Redwood Neuroscience Institute. He is currently a Researcher with Redwood Center for Theoretical Neuroscience, University of California at Berkeley. His thesis was published in the book *Sparse Distributed Memory* (MIT Press). His subsequent research includes binary spatter code, random indexing, and hyperdimensional computing.



Jan M. Rabaey (F'95) holds the Donald O. Pederson Distinguished Professorship with the University of California at Berkeley. He is currently the Electrical Engineering Division Chair, Berkeley. He is also the Founding Director of the Berkeley Wireless Research Center and the Berkeley Ubiquitous SwarmLab.

He has made high-impact contributions to a number of fields, including advanced wireless systems, low power integrated circuits, sensor networks, and ubiquitous computing. His current interests include the conception of the next-generation integrated wireless systems over a broad range of applications, and exploring the interaction between the cyber and the biological world.

He has been involved in a broad variety of start-up ventures. He is a member of the Royal Flemish Academy of Sciences and Arts of Belgium. He was a recipient of major awards, amongst which the IEEE Mac Van Valkenburg Award, the European Design Automation Association Lifetime Achievement Award, and the Semiconductor Industry Association University Researcher Award.