Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory

Hardware Optimizations of Dense Binary Hyperdimensional Computing: Rematerialization of Hypervectors, Binarized Bundling, and Combinational Associative Memory

MANUEL SCHMUCK, ETH Zürich
LUCA BENINI, ETH Zürich and Università di Bologna
ABBAS RAHIMI, ETH Zürich

Brain-inspired hyperdimensional (HD) computing models neural activity patterns of the very size of the brain's circuits with points of a hyperdimensional space, that is, with *hypervectors*. Hypervectors are D-dimensional (pseudo)random vectors with independent and identically distributed (i.i.d.) components constituting ultra-wide holographic words: $D = 10,000$ bits, for instance. At its very core, HD computing manipulates a set of seed hypervectors to build composite hypervectors representing objects of interest. It demands memory optimizations with simple operations for an efficient hardware realization. In this paper, we propose hardware techniques for optimizations of HD computing, in a synthesizable open-source VHDL library, to enable co-located implementation of both learning and classification tasks on only a small portion of Xilinx® UltraScale™ FPGAs: (1) We propose simple logical operations to *rematerialize* the hypervectors on the fly rather than loading them from memory. These operations massively reduce the memory footprint by directly computing the composite hypervectors whose individual seed hypervectors do not need to be stored in memory. (2) Bundling a series of hypervectors over time requires a multibit counter per every hypervector component. We instead propose a binarized *back-to-back* bundling without requiring any counters. This truly enables on-chip learning with minimal resources as every hypervector component remains binary over the course of training to avoid otherwise multibit components. (3) For every classification event, an associative memory is in charge of finding the closest match between a set of learned hypervectors and a query hypervector by using a distance metric. This operator is proportional to hypervector dimension (D), and hence may take $O(D)$ cycles per classification event. Accordingly, we significantly improve the throughput of classification by proposing associative memories that steadily reduce the latency of classification to the extreme of a single cycle. (4) We perform a design space exploration incorporating the proposed techniques on FPGAs for a wearable biosignal processing application as a case study. Our techniques achieve up to 2.39× area saving, or 2337× throughput improvement. The Pareto optimal HD architecture is mapped on only 18340 configurable logic blocks (CLBs) to learn and classify five hand gestures using four electromyography sensors.

CCS Concepts: •**Theory of computation → Random projections and metric embeddings;** *Online learning theory;* Active learning; •**Computer systems organization → System on a chip; Embedded hardware;** *Neural networks;* •**Hardware → Hardware accelerators;**

Additional Key Words and Phrases: Hyperdimensional computing, on-chip learning, FPGA, rematerialization, binarized temporal bundling, single-cycle associative memory, electromyography, biosignals

1 INTRODUCTION

Hyperdimensional (HD) computing [13, 15] is a brain-inspired computational approach based on the understanding that brains compute with patterns of neural activity that are not readily associated with scalar numbers. In fact, the brain's ability to calculate with numbers is feeble. However, due to the very size of the brain's circuits, we can model neural activity patterns with points of a hyperdimensional space, that is, with *hypervectors*. When the dimensionality is in the thousands, operations with hypervectors create a computational behavior with remarkable properties [11]. HD computing builds upon a well-defined set of highly parallel operations with

random hypervectors, is extremely robust in the presence of failures, and offers a complete computational paradigm that is easily applied to many learning applications [29]. Examples include analogy-based reasoning [14], latent semantic analysis [16], language recognition [10, 32], text classification [23], speech recognition [8, 34], physical activity prediction [35, 36], robot learning by demonstration [19, 24], and several biosignal processing tasks such as electromyography (EMG) [18, 21, 22, 28], electroencephalography (EEG) [31, 33], electrocorticography (ECoG) [1], and in general ExG [30].

In contrast with traditional approaches, learning in HD computing is fast and computationally balanced with respect to classification by reusing the same algorithmic and architectural constructs for both modes of operation. Its learning is not iterative and thus requires far fewer operations than other approaches (see [30] for an overview). Another advantage of HD computing is the simplicity of its basic operations, which is an important factor for energy efficiency. For instance, HD computing achieves 2× lower energy consumption at iso-accuracy when compared to a highly-optimized support vector machine (SVM) with fixed-point operations on a commercial embedded ARM Cortex M4 processor for an EMG classification task [22]. In what follows, we brief these basic operations.

At its very core, HD computing is all about generating, manipulating, and comparing hypervectors as ultra-wide words. As the first step, D-dimensional hypervectors are initially generated with independent and identically distributed (i.i.d.) components. Second, these *seed* hypervectors are manipulated to construct composite hypervectors, as richer representations, with componentwise arithmetic operations by needing to communicate with only a local component or an immediate neighbor. By using dense binary codes for hypervectors [12], the arithmetic operations simply involve bitwise XOR, shift (or rotate), and majority gates [32]. Finally, the constructed hypervectors are compared for *similarity* using a distance metric whose computation involves a reduction operator proportional to the hypervector dimension (D) [9, 32]. See Section 2 for details. These operators—at the basis of both learning and inference (Section 3)—demand a memory-centric architecture for efficient and local ultra-wide word processing. Emerging nanotechnologies with dense 3D integration can provide a natural fit [20, 39, 40].

In this paper, we propose hardware techniques to optimize the aforementioned operators to build an efficient acceleration engine on an FPGA. As a result of our hardware optimizations (Section 4), we provide a synthesizable VHDL library[1] of fully configurable modules exploring trade-offs between area and throughput of the operators. Our contributions are as follows:

(1) We propose a generic hypervector manipulator (MAN) module as a fully combinational logic consisting of OR-XOR gates and preprogrammed connections. The MAN module substitutes the expensive memory storage for maintaining seed hypervectors with cheaper logical operations to *rematerialize* them. Hence, representations of composite hypervectors are constructed directly by rematerializing the seed hypervectors as a consequence of reusing the generic MAN modules that form a combinational network architecture without requiring any memory storage.

(2) The arithmetic operations of HD computing with dense binary code exhibit their simplest form by performing local and bitwise operations on binary components. This however does not hold for the majority gate when it is applied to bundle a series of hypervectors over time, i.e., among different training examples. Implementation of the majority gate requires to maintain intermediate (i.e., partially bundled) hypervector representation using a set of D multibit counters—every counter counts the number of 1s in a specific dimension. We rather reuse the generic MAN module that replaces the multibit hypervector components with binarized hypervector components by incrementally applying an approximate majority gate for every training example. Such a

[1]The library is open-source and available at: github.com/eardbi/hd-vhdl-library

binarized back-to-back bundling enables the representational system to continuously stay in the binary space that is essential for efficient on-chip learning during the course of online learning.

(3) The common denominator of all architectures of HD computing is the extensive use of distance computation in the associative memory that typically takes $O(D)$ cycles per every event of classification. We propose associative memories to significantly reduce the classification latency to single cycle.

(4) We perform a design space exploration of our library modules for an application which recognizes hand gestures from four EMG senors (Section 5). It shows that functionally equivalent HD architectures can be composed achieving up to 2.39× area saving, or 2337× throughput improvement. The Pareto optimal HD architecture is fully synthesized on only 18340 CLBs of the Xilinx® UltraScale™ FPGAs, and shows simultaneous 2.39× area and 986× throughput improvements compared to a baseline HD architecture.

2 BACKGROUND

HD computing is rooted in the observation that key aspects of human memory, perception and cognition can be explained by the mathematical properties of hyperdimensional spaces, and that a powerful system of computing can be built on the rich algebra of hypervectors [13]. A further motivation is the fact that brains compute with *patterns of neural activity* that are not readily associated with numbers. In fact, recognizing the very size of the brain's circuits, we can model neural activity patterns with points in a hyperdimensional space. Computing in hyperdimensional space is understood partly in terms of the linear algebra and probability of artificial neural nets, and partly in terms of the abstract algebra and geometry of hyperdimensional spaces. Groups, rings, and fields over hypervectors become the underlying computing structure, with permutations, mappings, and inverses as primitive computing operations, and with randomness as a way to label new objects and entities.

Hypervectors are D-dimensional, *holographic*, and (pseudo)random with i.i.d. components. It means that the contained information in a hypervector is distributed equally over all D components: neither a component nor a subset of them have a specific meaning, hence the information degrades in relation to the number of failing components *irrespective of their position.* The high dimensionality yields a huge number of different, nearly orthogonal hypervectors in such space [11]. They can be mathematically manipulated for solving cognitive tasks, e.g., Raven's progressive matrices [4], analogical reasoning [14], and practical learning and classification tasks [7, 8, 10, 18–24, 28, 29, 31–36, 40]. Examples of such computing include Holographic Reduced Representation [25, 26], Binary Spatter Code [12], Multiply-Add-Permute architecture [5], Random Indexing [16], and Semantic Pointer Architecture Unified Network [3], collectively referred to as Vector Symbolic Architecture [6]. They differ in the type of components, and the types of operations, however, the key properties are shared by hypervectors of many kinds, all of which can serve as the computational infrastructure. To ease the hardware realization, we focus on Binary Spatter Code (BSC), where the components of hypervectors are binary and dense, meaning the probability of having a 1 or a 0 is equal ($p = 1/2$) [12].

2.1 Measure of Similarity

Using BSC, $\{0,1\}^D$, the similarity between two hypervectors is given by the number of components at which they differ, the so-called *Hamming distance*. We use the normalized version of this metric by dividing by D denoted as: $d(X,Y) : \{0,1\}^D \times \{0,1\}^D \rightarrow [0,1]$ to express the distance on a real scale of 0 to 1. Figure 1 shows the normalized Hamming distance distribution of hypervectors in D-dimensional spaces where $D \in \{100, 1000, 10000\}$. As we go to higher dimensions from $D = 100$

to $D = 10,000$, we observe an outstanding property: most points are $D/2$ bits apart from each other, which yields a normalized Hamming distance of $d \approx 0.5$, and stands for two nearly orthogonal hypervectors. This stems from the binomial distribution for $p = 1/2$ and $n = D$, where $D/2$ is the mean. Correlated hypervectors yield $d \approx 0$ whereas $d \approx 1$ implies anti-correlation [13].

Orthogonality Condition. When approximating the discrete binomial distribution with the continuous normal distribution, its standard deviation is $\sqrt{D}/2$. According to the normal distribution, $\approx 68.2\%$ of the space lies within one standard deviation from the mean or within $\sqrt{D} \pm 1$ standard deviations from a point in the hyperdimensional space [11]. If we increase the range to 6 standard deviations, already $\approx 99.9999998\%$ of the space lies within that range. This marks our orthogonality threshold as $d_{orthogonality} = \frac{\sqrt{D} \cdot (\sqrt{D} \pm 6)}{2 \cdot D}$ which states that with a chance of $\approx 99.9999998\%$ two random hypervectors exhibit a normalized Hamming distance in the aforementioned range. For $D = 10,000$ this yields a range between 0.47 and 0.53 [11]. In other words, almost all the space lies at approximately the mean distance of [0.47,0.53] from a chosen random point; this implies that for any significant deviation from distance 0.5, the distribution quickly becomes very sparse.

2.2 HD Arithmetic Operations

The HD algorithm starts by choosing a set of seed hypervectors as initial items. They are stored in a so-called item memory (IM) as a symbol table or dictionary of all the hypervectors defined in the system. They stay fixed throughout the computation, and they serve as seeds from which further representations are made. HD computing builds upon a well-defined set of operations with the seed hypervectors [13]. These arithmetic operations are used for encoding and decoding patterns. The power and versatility of arithmetic derives from the fact that addition and multiplication form an algebraic field, and permutation of hypervector components takes it beyond both arithmetic and linear algebra.

Addition (Bundling). The sum of binary hypervectors is defined as the componentwise majority function (also called the median operator) with ties broken at random. This means, when adding an even number of hypervectors, in case of disagreement for a component (equal number of 1s

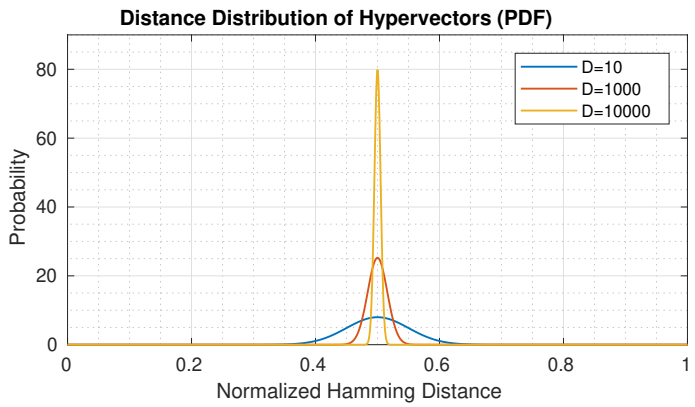


Fig. 1. Normalized Hamming distance distribution of hypervectors in D-dimensional spaces. As the dimensionality increases, the standard deviation $(1/(2\sqrt{D}))$ of the normalized distance distribution between two random hypervectors decreases. This implies that the probabilty of two random hypervectors lying about $d \approx 0.5$ apart from each other increases with the dimension D.

and 0s), the majority is randomly chosen. It is denoted as $A \oplus B$. The sum of two hypervectors stores information from both hypervectors, due to the mathematical properties of vector addition, therefore the operation is also called *bundling*. Bundling two hypervectors yields a hypervector which is similar to both of them, hence it is well-suited for representing sets or multisets. However, when breaking ties at random, the bundling operation becomes non-causal. Furthermore, the bundling is commutative but not associative and is only approximately invertible.

Multiplication (Binding). The product of two binary hypervectors is defined as the componentwise XOR or "addition modulo 2", and is denoted as $A \otimes B$. The resulting hypervector is dissimilar (orthogonal) to both its constituent hypervectors, which is why multiplication is well-suited for *binding* two hypervectors. Binding is commutative, associative and distributes over bundling. The operation can be inverted and also preserves distances between hypervectors, meaning two similar hypervectors (after binding) are mapped to equally similar ones.

Permutation. The third operation, denoted $\rho(A)$, is the permutation operation, which shuffles a hypervector's components by rotating it in space. It is implemented as a cyclic shift by one position. Permuting a hypervector produces a dissimilar, pseudo-orthogonal hypervector, which can be exploited to bypass the commutativity of the other operations. This is crucial when storing sequences, where e.g., a-b-c should be distinguishable from b-c-a. Permutation is invertible and preserves distances. It distributes over both bundling and binding.

These three operations can be combined to encode structures such as variable/value records, sequences, and lists—essentially any data structure. For example, let us consider three variables x, y, z and their values a, b, c. Each of them is mapped to a (random) hypervector X, Y, A, B etc., which are stored in the IM. Then, the entire of a record is encoded to a single hypervector by binding each value to its variable and bundle them to form the holistic record: $R = (X \otimes A) \oplus (Y \otimes B) \oplus (Z \otimes C)$. To find the value of x, we unbind the record with the inverse of X (which is X itself), $\tilde{A} = X \otimes R$ which gives us a hypervector $\tilde{A}$ as noisy version of A. After comparing it with the hypervectors that are stored in the AM, we find A to be the most similar one (i.e., the lowest Hamming distance), and thus the sought value.

3 LEARNING AND CLASSIFYING MULTICHANNEL BIOSIGNALS WITH HD COMPUTING

In this section, we describe how to use HD computing for learning and classification tasks. We focus on wearable biosignal processing applications with multichannel noisy sensors for which HD computing achieves faster training and lower energy consumption and memory than SVMs [1, 22]. One application example includes recognizing hand gestures from a stream of EMG sensors to control a prosthetic device [22, 28]. The performance of HD computing however depends on good design of a network architecture that demands a reconfigurable (FPGA) fabric to efficiently arrange the HD primitive operations based on the given task. We present a generic architecture to project multichannel sensory inputs from original representation to hyperdimensional space, where the arithmetic operations are combined to learn and classify examples. While this paper focuses on EMG signals, other streaming multichannel sensor data such as ECoG [1], EEG [31, 33], ExG [30], speech [8, 34], smell [7] can be equally applicable.

The dataset [28] used in this paper is based on a four-channel EMG data acquisition, among five subjects, for the most common hand gestures in daily life. The selected gestures are: closed hand, open hand, 2-finger pinch, point index, and the rest position. The recording is composed of 10 trials of every gestures three seconds each. We use 25% of this dataset for training that can be performed

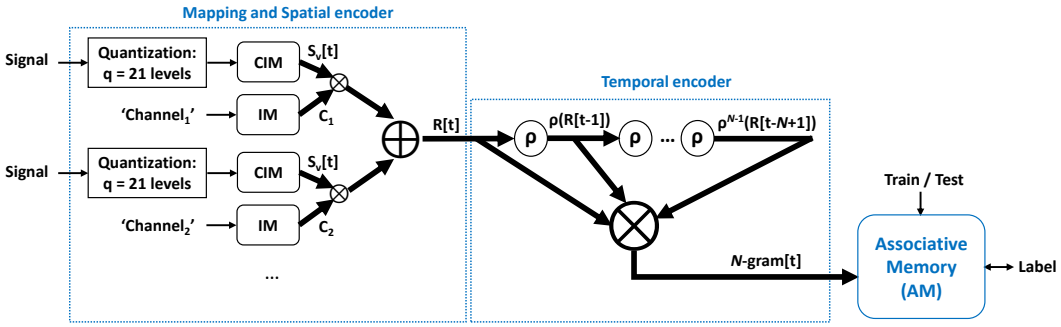


Fig. 2. Example of an HD architecture for hand gestures learning and classification from EMG biosignals.

online. The gestures are sampled at 500 Hz, followed by a low pass filter, and an envelope signal extraction; [28] provides further details about the setup.

3.1 HD Architecture

As shown in Figure 2, an HD architecture consists of three main modules: mapping and spatial encoder, temporal encoder, and associative memory. The mapping and encoding modules intend to capture information that can be extracted from the inputs (i.e., the enveloped EMG signals), into a hypervector representing a gesture. Gesture hypervectors, extracted from various trials, are bundled to form a *prototype* hypervector representing a *class* of gestures. The associative memory (AM) stores a prototype hypervector for every class, which contains the encoded information of all labelled inputs during the training phase. During inference, classifying input data is carried out by comparing the unlabelled encoded hypervectors with all stored prototype hypervectors, and returning the label of the most similar one.

3.2 Mapping and Spatial Encoder

First, the analog EMG signals have to be quantized to q discrete levels, where q indicates the resolution of the signal. In analogy to the record example in the previous section, the different EMG channels represent the variables or fields, and the discretized signals represent the values of the variables. All channels are treated as separate and independent, therefore we allocate each one a random and thus orthogonal hypervector, which are fixed throughout the computation in the IM: $C_1 \perp C_2 \perp C_3 \ldots \perp C_n$. Figure 3a shows the IM with four channels.

Each of the channel variables has a corresponding value, i.e., the discretized signals. When mapping quantities from the discrete number space to the hypervector space, we want to retain their similarity: e.g., with a resolution of $q = 21$ levels, a value of 5 is only slightly larger than a value of 4, hence their allocated hypervectors shall not be orthogonal [28]. For mapping such quantized or even continuous values into hypervectors various techniques can be used including thermometer codes, locality-sensitive hashing, or generally, random projection [27]. We use the following simple method to map the values to a continuous vector space. A random seed hypervector is taken for the smallest value and the hypervectors for the other levels are generated such that they are gradually further away from the seed up to the largest value, whose hypervector is orthogonal to the seed. We can accomplish this by randomly choosing $D/2$ components of the seed and split them into $q-1$ groups which equally contain $(D/2)/(q-1)$ components. The hypervectors are then generated from the seed by taking one group after the other and flipping their components. For the last hypervector, exactly $D/2$ components are flipped, making it orthogonal to the seed. These

generated *signal* hypervectors are denoted by S_v where $v \in [0, q-1]$, that are stored in the so-called *continuous item memory* (CIM). Figure 3b illustrates a CIM with $q = 21$: $d(S_n, S_{n+i}) = 0.5 \cdot \frac{i}{q-1}$ hence $d(S_0, S_{q-1}) = 0.5$.

As mentioned in Section 2.2, we aim to bind the values to their variables and bundle them to form a holistic record (R) to capture spatial information between all channels. The signal hypervector of a channel at time t, is denoted by $S_v[t]$ where $v \in [0, q-1]$. Hence, a record is computed for a given time-aligned sample of all channels: $R[t] = (C_1 \otimes S_v[t]) \oplus (C_2 \otimes S_v[t]) \oplus (C_3 \otimes S_v[t]) \oplus (C_4 \otimes S_v[t])$. As shown in Figure 2, this record contains the signal information of all channels, while distinguishing the source of the signals (i.e., the channels).

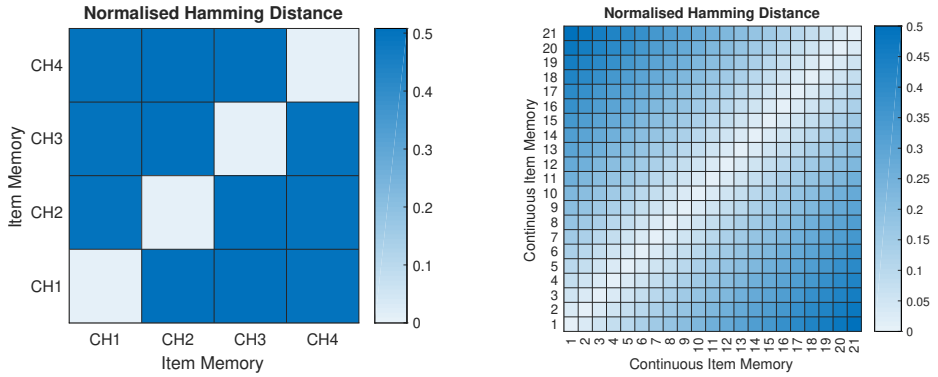
3.3 Temporal Encoder

We can encode sequences by using the permutation operation ρ. Hence, we can capture not only the spatial correlation across the channels, but also the temporal correlation between subsequent samples. We call a sequence of N record hypervectors as an N-gram hypervector.

As already mentioned, a sequence of hypervectors can be encoded uniquely by permuting the hypervectors before binding them. The sequence is encoded by rotating the first spatial record $N-1$ times, the second $N-2$ times, and the $(N-1)$th only once. The Nth hypervector is untouched (not permuted). These new hypervectors are finally bound to an N-gram (see Figure 2). For large N-grams, this becomes: $N\text{-gram}[t] = \prod_{i=0}^{N-1} \rho^i(R[t-i])$. An N-gram contains the spatial information of N subsequent samples with different timestamps, making it a *spatiotemporal* hypervector.

3.4 Learning and Classification in Associative Memory

In a typical training setting, a set of labelled examples is provided per every class. By encoding the sensory data, a current gesture example is represented by an $N\text{-gram}[t]$ hypervector. The HD architecture learns from these N-gram hypervectors that are produced over time. A number of N-gram hypervector examples (e.g., k) with the same label are bundled to produce a prototype hypervector representing the class of interest: $P_{\text{Label}_i} = N\text{-gram}_{\text{Label}_i}[t] \oplus \ldots \oplus N\text{-gram}_{\text{Label}_i}[t+k]$. Once training is done, the binarized prototype hypervectors are stored in the AM as *learned patterns*. This *temporal* bundling of N-grams over the course of training requires D counters and thresholders to implement the majority function.



(a) The IM with four channels

(b) The CIM with $q = 21$ levels

Fig. 3. Similarity, depicted as a heatmap, between hypervectors of the IM (a) and the CIM (b).

As soon as the AM is trained for each class, it can identify the corresponding class of an unlabelled N-gram, which is called a *query hypervector*. More specifically, the AM computes the Hamming distance between the query hypervector and each of its prototype hypervector. It then selects the highest similarity and returns its associated label. As shown in Figure 2, the same construct is reused during inference, the only difference is that during training the prototypes are written into the AM while during inference they are read and compared with the query.

4 HARDWARE OPTIMIZATIONS OF DENSE BINARY HD COMPUTING

In this section, we present the main contributions of the paper. We present our techniques to optimize hardware realization of HD computing suitable for CMOS fabrics. HD computing demands a large amount of bits to be stored for each data item that further poses a memory bandwidth issue, for instance the IP RAMs of FPGAs are optimized for usually no more than 72 bits in parallel [41]. Storing or loading one hypervector in this fashion would require hundreds of cycles. Accordingly, optimizing the architecture of HD computing should focus on minimizing the number of stored hypervectors. Furthermore, the bitwise operations need to be kept as simple as possible, since they are replicated over the whole dimension of a hypervector. Most architectural constructs are shared among various HD classifiers and thus the optimizations virtually concern all HD computing applications.

As a result of various hardware optimizations, we introduce a synthesizable VHDL library of fully configurable modules which comprises different implementations. The VHDL library consists of interchangeable modules including three types of spatial encoder, two types of temporal encoder, and three types of AM, that are listed in Table 1. A functioning HD architecture can be configured by connecting one type of each of the modules in series. The modules operate independently and pass hypervectors after synchronizing via handshake signals. They all differ greatly in area and throughput, where the number of cycles needed to process a data item (CPDI) has the biggest influence on throughput. Table 1 shows the CPDI for the different modules.

4.1 Mapping Multichannel Sensory Inputs

Mapping the input data of more than one channel to the hyperdimensional space can be done in a parallel fashion as shown in Figure 2. The required memory for the IM and the CIM is $n_c \times q \times D$ where n_c is the number of channels, q is the quantization of input signal, and D the hypervector dimension. For the EMG task (see Figure 2) this would be equal to 840 kbits to only store the seed hypervectors. This poses limitations when a large number of channels [21] or input quantization is used. A first step is to trade the high throughput against a smaller memory footprint by sharing the resources.

Table 1. Cycles per data item (CPDI) orders of the different library modules.

(a) Spatial encoder modules.

Module	CPDI
LUT	$O(1)$
CA	$O(n_{\text{channels}})$
MAN	$O(n_{\text{channels}})$

(b) Temporal encoder modules.

Module	CPDI
BC	$O(1)$[a]
B2B	$O(1)$

[a] This holds only for inference. During training, the order is of the number of training samples.

(c) Associative memory modules.

Module	CPDI
BS	$O(D)$
CMB	$O(1)$
VS	$O(n_{\text{classes}})$

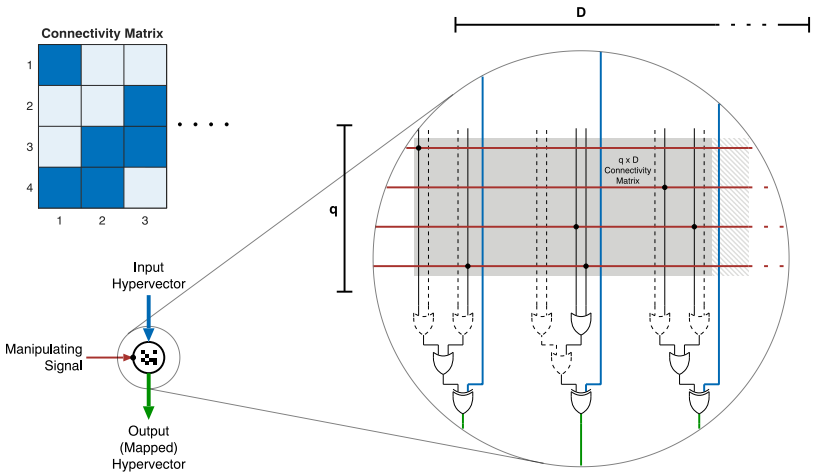


Fig. 4. The hypervector manipulator (MAN) module and its symbol representation. The connectivity matrix serves as an example. The other symbols used throughout this paper can be found in Figure 5.

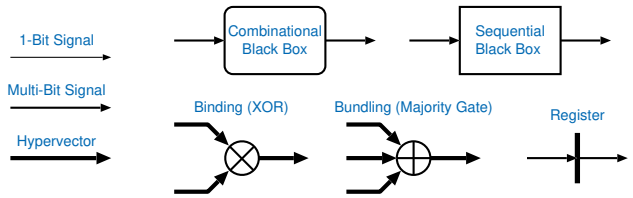


Fig. 5. The symbols used for schematic drawings throughout this paper.

4.1.1 Rematerialization: Replacing CIM with MAN. A single CIM implemented as a lookup table requires $q \times D$ bits of storage. To reduce this memory footprint we can exploit the holographic nature of HD representation: the individual bits in a hypervector do not represent anything. What is important is the relation or similarity between two hypervectors. A hypervector can be altered or "manipulated" to a different hypervector by switching certain bits as a function of the similarity that we want to establish. For example, to obtain an orthogonal hypervector, we have to switch half of its bits (which ones does not matter), whereas to obtain a similar hypervector, we only switch a (small) portion of the bits (see Section 2.1).

Manipulating hypervectors in a controlled manner can replace complex constructs throughout the whole architecture. For this purpose, a generic hypervector manipulator (MAN) module is designed (Figure 4), which can be configured in depth and width, and is fixed by a *connectivity matrix*, which determines the connections between wires. An example connectivity matrix used for mapping is shown in Figure 7.

Every cell of the connectivity matrix affects, whether a certain bit of the input hypervector can be switched by a bit (or even several bits) of the *input manipulator*. The MAN module is a simple combination of OR and XOR gates. If a cell (m, n) of the connectivity matrix is set to 1, the m-th bit of the input manipulator can affect the n-th bit of the input hypervector: when the m-th bit of the input manipulator is logically high it toggles the n-th bit of the input hypervector. The number of 1s in a row of connectivity matrix also represents how dissimilar the output hypervector will be to

the input hypervector when the input manipulator bit of that row is logical high: the fewer the number, the more similar.

As described in Section 3.2, "close" input values are mapped to similar hypervectors using a CIM. This CIM can be replaced by a MAN module that produces similar hypervectors according to the input value. First, the quantized input value in binary representation is mapped to an *s-hot* representation (by e.g., a small lookup table), where s is the input/signal value (see Figure 6). This s-hot code serves as the input manipulator, and gradually switches more and more bits of a *seed* input hypervector as the input value goes higher, and eventually produces an orthogonal hypervector when all q bits are hot (q is the quantization). This allows to rematerialize desired hypervectors from a seed by keeping track of the input value.

Which bits are switched is chosen randomly (without the possibility to choose a bit twice), only the number of bits per "input quantum"—represented by a row in the connectivity matrix—is determined. It is equal to $D/2/(q-1)$. Moreover, every input hypervector bit can only be switched by one input manipulator bit. This results in a MAN module containing only XOR gates. The input hypervector that is manipulated is a constant seed hypervector (S_0) which represents the lowest input value, or 0-hot. This seed hypervector is simply hardwired connections to source and ground. Summing up, the whole continuous item memory, or CIM, is replaced with a rather small s-hot lookup table memory of size $q \times q$, some wires, and $D/2$ XOR gates.

4.1.2 Reproducing IM with Cellular Automata. As mentioned in Section 3.2, we account for the spatial multichannel information to determine which channel the data originated from. This is done by binding a channel hypervector, that is unique for every channel, with the signal hypervector. The channel hypervectors are typically stored in the IM with a memory of size $n_c \times D$. When mapping the input data in the parallel fashion, the IM can be replaced by hard wires tied to source and ground since the channel hypervectors are constant. However, with the serial mapping, they need to be stored in the IM.

One way to replace the IM is by using a one-dimensional cellular automaton (CA) with a *neighborhood* of 3, applying *rule 30* [38]. This rule exhibits chaotic behaviour that is well-matched to produce a sequence of (quasi-)random hypervectors. When using a CA with D cells and a random hypervector as initial state, it generates (quasi-)random and orthogonal hypervectors every cycle (see Figure 8). By resetting the CA registers, the same sequence can be reproduced (i.e.,

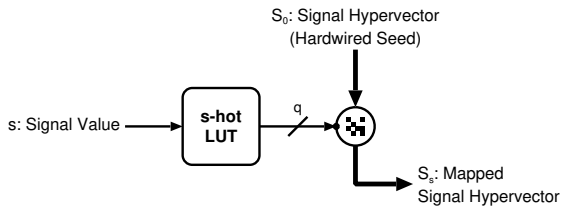


Fig. 6. Data dependency graph of the MAN module to replace a CIM.

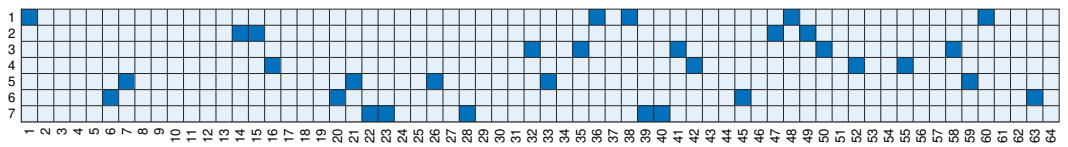


Fig. 7. Example of a connectivity matrix to map an input with $q = 8$ to a hypervector of dimension $D = 64$.

rematerialized) over and over. This allows us to replace the IM (see Figure 9) by only defining the initial state of the CA as a seed hypervector and letting it generate the other orthogonal hypervectors[2] for the rest of the channels. Thanks to the chaotic behaviour of the CA, this approach works for virtually any number of channels: clocking the CA for 500 cycles produces the channel hypervectors for 500 channels only from the initial state hypervector (see Figure 8).

Although the gate logic required for each cell in CA is quite simple—only consisting of 3 inverters, 4 two-input AND gates and 2 two-input OR gates—it is still replicated D times. When looking for a solution to generate orthogonal hypervectors at relatively low costs, CA are an excellent choice, whereas when looking for an optimal solution for spatial encoding, further improvement can be done as described in the following section.

4.1.3 Replacing Both IM and CIM with MAN. The MAN module in Section 4.1.1 can also be applied to replace the IM. Instead of storing the channel hypervectors, their patterns can be incorporated in the connections of the MAN module. The connectivity matrix in this case is identical to an IM and has an average of $D/2$ 1s per row as shown in Figure 10. Feeding signal hypervector to the second MAN module and setting one bit of its input manipulator logical high at a time yields the same outcome as binding the signal hypervector with a channel hypervector (see Figure 13c).

The second MAN module (replacing the IM) requires more gates due to its dense connections than the first one (replacing the CIM). The chance that a channel hypervector switches a certain bit is 0.5 (the probability of having a 1 in a component), hence this yields an average of $n_c/2$ connections per column in the connectivity matrix (see Figure 10) which have to be OR-ed before

[2]In case the "randomness" of rule 30 is not enough, the neighbourhood can be extended to form a more complex CA as in [37].

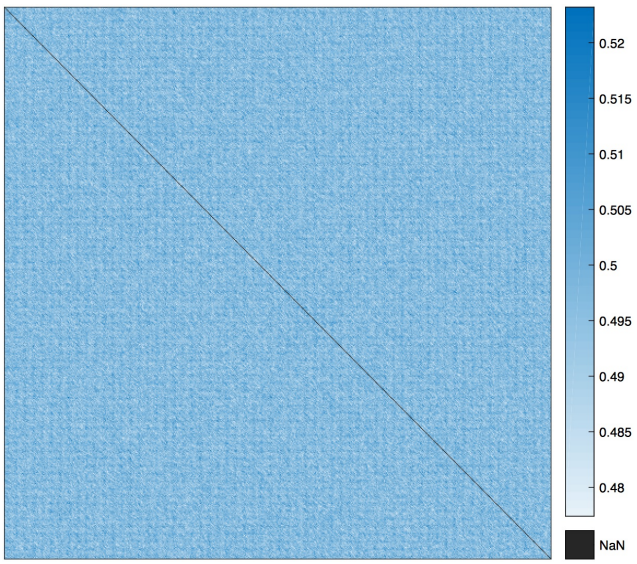


Fig. 8. A heat map showing the orthogonality (normalized Hamming distance) between hypervectors produced by the CA (rule 30) over 500 cycles. Each dot (x, y) on the graph shows the Hamming distance between the hypervector produced in cycle x and the one produced in cycle y. As shown in the minimum and maximum values of the color scale on the right, the orthogonality condition from Section 2.1 is met.

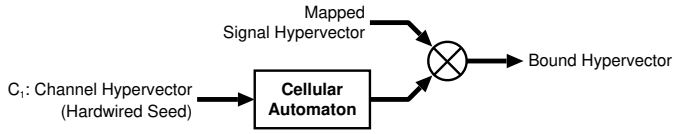


Fig. 9. Data dependency graph of the spatial encoding architecture with the cellular automaton (CA).

going into the XOR gate. This operator per hypervector bit is replicated D times to replace the whole IM.

4.2 Spatial Encoding

The hypervectors that contain information of the input signal values and the channels should be bundled in the spatial encoder. In Section 2.2, the bundling operation is characterized as a method to store the information of multiple hypervectors in a single hypervector, called a record, which is similar to all of the input hypervectors. The information of a hypervector is contained in another as long as they do not violate the similarity condition (Section 2.1). Here, we investigate how well this task is accomplished by the majority function, and how it can be implemented in hardware and whether there are other approaches to achieve the same goal.

4.2.1 The Three Problems of the Majority Function.

The Majority Function of an Even Number of Inputs. The majority function (or vote) for binary inputs is self-explanatory and only yields a clear result with an odd number of inputs. This is why the concept of *braking ties at random* is introduced [13], which makes the operation non-causal for an even number of inputs and is identical to bundling an additional random (and thus orthogonal) hypervector into the record. Therefore, two records, that are supposed to be equal, become (slightly) dissimilar. Instead of "wasting" said similarity, an additional hypervector can be introduced, that contains useful information, to break the ties. In the case of bundling hypervectors from multichannel, useful information could come from an additional channel. If this is not an option, we can synthetically create that information. It should be "useful" in the sense, that it is unique for the given input and also causal. Binding a constant hypervector would lead to all output hypervectors being slightly similar to each other even if they are supposed to be orthogonal. Instead, by simply binding any two of the input hypervectors (see Figure 11), we can create an *additional feature*, which represents the input data and is useful as stated before.

Unfairness of the Majority Function. Bundling hypervectors with the majority vote does not yield their mean hypervector but strongly tends to the majority of the hypervectors. This means, if we want to store the information of e.g., three hypervectors, where two of them are equal and the

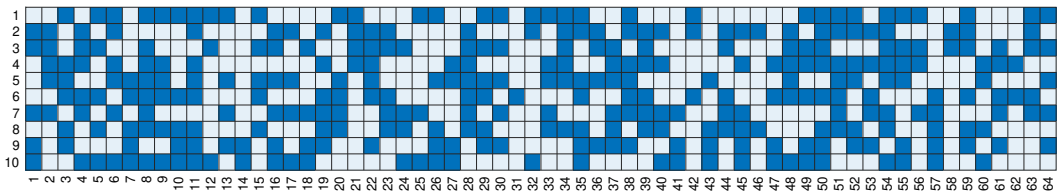


Fig. 10. Example of a connectivity matrix to replace the IM for a hypervector dimension $D = 64$ bits and 10 input channels.

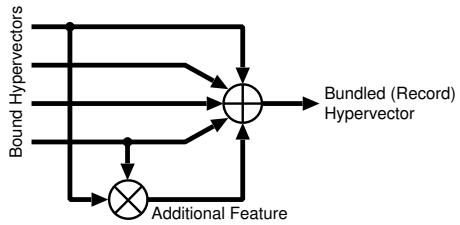


Fig. 11. Data dependency graph of the bundling with even inputs using an additional feature.

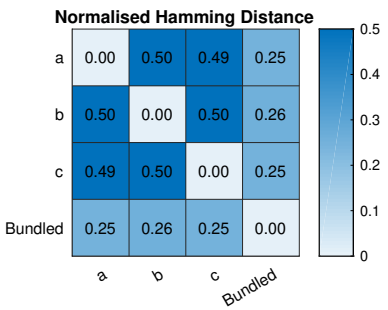
other is orthogonal, the information of the latter is lost entirely (see Figure 12b). The same situation occurs when bundling two sets of hypervectors to one record, where the sets are dissimilar to each other, but similar within. The smaller set will not be recalled at all. In Section 4.5.1, another bundling approach will be presented, which is completely fair in this case.

When bundling only orthogonal hypervectors, this problem does not occur and the majority function is fair (Figure 12a). This rises the question of the "capacity" of the bundling operations (see Section 4.5.2).

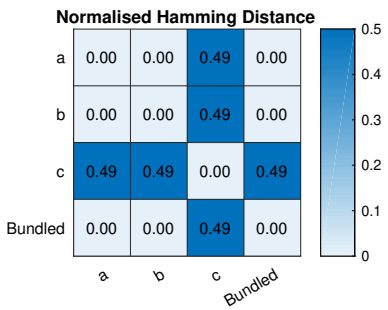
Lack of Associativity. When attempting to implement the bundling operation, one quickly comes across a mathematical property that is necessary to conduct an operation in an iterative manner: *associativity*. The majority function lacks this property, meaning a set of hypervectors can only be bundled altogether, but not step by step: $a \oplus b \oplus c \neq (a \oplus b) \oplus c$. Fortunately, one is not tied to mathematical properties, when it comes to the algorithmic and architectural implementation of an operation. The workaround lies in storing the current vote over an iteration.

4.2.2 Bidirectional Saturating Counters as a Hardware Implementation of the Majority Function. A naive approach to store the current majority vote would be to count the vote for 1s and 0s with two separate counters and compare their values to get the majority. This would require a memory of $2 \cdot D \times \lceil \log_2(n_c + 1) \rceil$ which, for only $n_c = 4$ input channels in our EMG task would already yield 60,000 bits.

The two counters can be combined to a single one that counts up or down depending on the value of the current bit to reduce the memory to $D \times (\lceil \log_2(n_c + 1) \rceil + 1)$. The next big improvement is made by exploiting the random nature of orthogonal hypervectors. Observing a single component of the input vectors, the probability of a long sequence of either 1s or 0s is small, implying the



(a) $a \perp b \perp c$



(b) $a = b \perp c$

Fig. 12. Similarity between three hypervectors and the bundled hypervector $a \oplus b \oplus c$ using majority vote.

counter usually does not have to count all the way up to the maximum possible vote, but stays within a certain range. Taking a counter with a fixed width and forcing it to saturate whenever it would traverse that range, assures that the vote is not passed to the other extreme, which occurs when letting it wrap around.

With this approach, the maximum accuracy of the majority function can be reached with a certain width of the counter. For a hypervector dimension $D = 10,000$ the maximum width is 5 bits resulting in a memory of $50,000$ bits which is independent from the number of hypervectors to be bundled, and is maximally memory-saving for a large number of input channels. The downside is the complexity of a saturating counter. Due to the orthogonality of the hypervectors for bundling inside the spatial encoder, the saturating counter method is the preferred approach because of its large capacity and moderate complexity.

4.3 Library: Spatial Encoder Modules

The following library modules emerged from the optimizations in Section 4.1 and 4.2:

- *LUT.* A purely combinational, LUT-based spatial encoder architecture. This is the starting point for optimizations and was described in [28]. See Figure 13a.
- *CA.* A sequential spatial encoder architecture, where the IM is reproduced by a cellular automaton (CA) as described in Section 4.1.2. The bound hypervectors are bundled by a block of bidirectional saturating counters as described in Section 4.2.2. See Figure 13b.
- *MAN.* A sequential spatial encoder architecture, where the IM is "hardwired" in a manipulator's connectivity matrix as described in 4.1.3. The same bundling method as in the CA module is used. See Figure 13c.

A summary of the CPDI of all library modules can be found in Table 1.

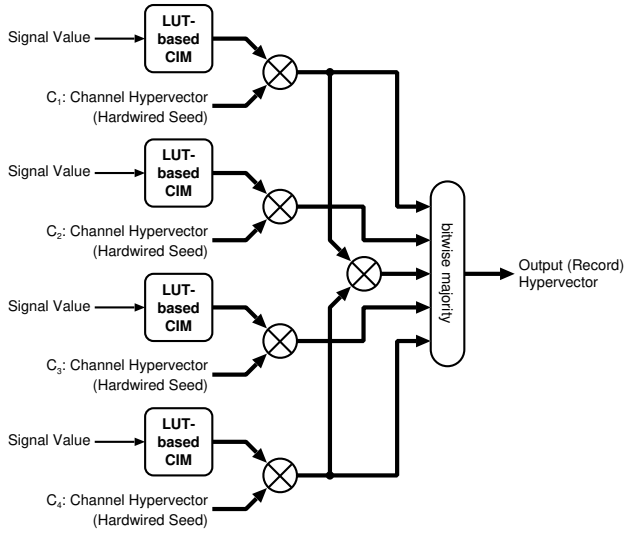
4.4 Temporal Encoding

As mentioned in Section 3.3, the temporal encoder considers consecutive samples over time. This is done by rotating and binding the record hypervectors to an *N-gram* hypervector: $N\text{-gram}[t] = R[t] \otimes \rho(R[t-1]) \otimes \rho^2(R[t-2]) \otimes \cdots \otimes \rho^{N-1}(R[t-(N-1)])$.

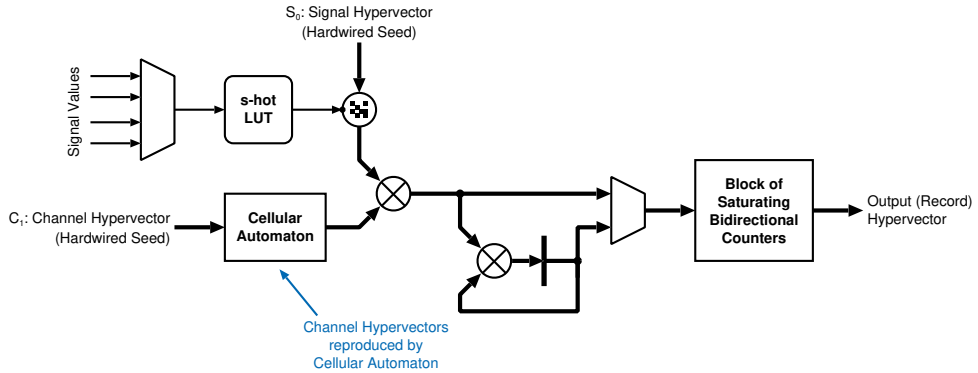
In order to deliver a new N-gram every cycle, the records of the last $N-1$ cycles have to be kept in memory. For this, the first record is rotated and stored. In the next cycle it is again rotated and stored, while the new record is rotated and stored where the last record was stored, and so on. In parallel, the current record is bound with all stored records and a valid N-gram is produced every cycle (see Figure 14).

4.5 Bundling N-gram Hypervectors

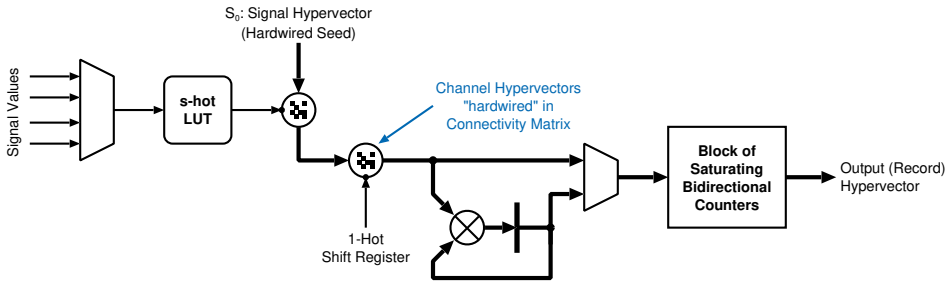
All the modules that are described so far in this section form an HD projection along with a *spatiotemporal* encoder. This also constitutes a shared construct between learning and inference because the hypervectors that are produced at the output of spatiotemporal encoder (i.e., the N-gram hypervectors) contain all the information about the event of interest (e.g., a gesture) for training or classification. The AM is another part of the shared construct; however, the output of encoder queries the AM during classification while updates it during training. For training a certain class, its N-gram hypervectors need to be bundled before writing into the AM. Different examples of a gesture are usually encoded to similar N-gram hypervectors, since they belong to the same class. This calls for a bundling method that does not require the capacity of an accurate majority function implemented with the complex saturating counters.



(a) Lookup table (LUT): Parallel encoder with LUT-based CIMs and no IMs.



(b) Cellular automaton (CA): Sequential encoder with CA (replacing IM), and MAN module (replacing CIM).



(c) MAN: Sequential encoder replacing both CIM and IM with two cascaded MAN modules.

Fig. 13. The spatial encoder architectures available in the library.

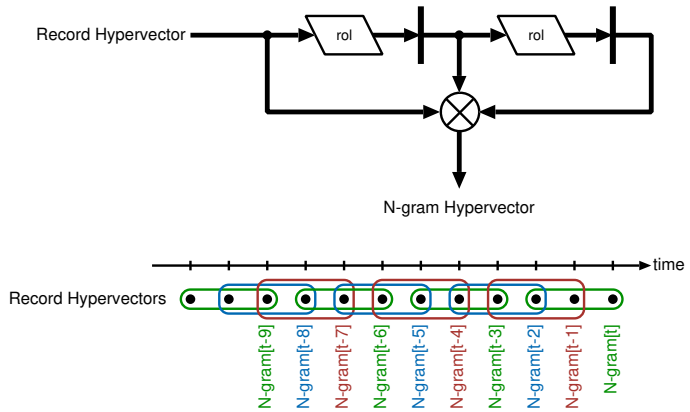


Fig. 14. Top: Data dependency graph of the window-shifting N-gram encoder. Bottom: Depiction of the timeline of generated N-grams.

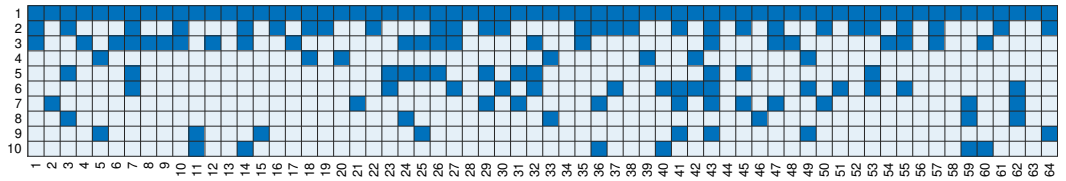


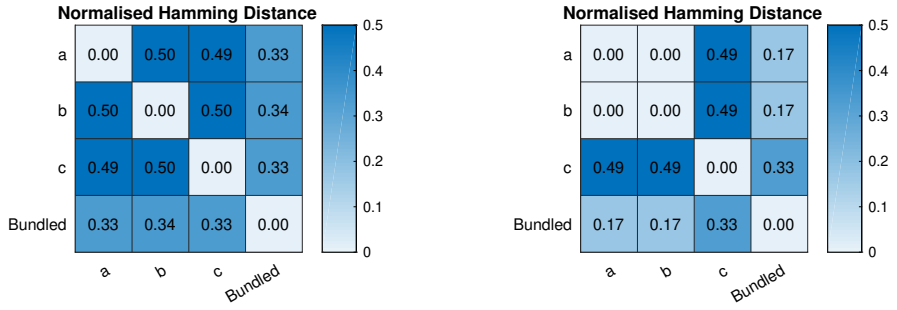
Fig. 15. Example of a connectivity matrix to bundle 10 hypervectors of dimension $D = 64$ iteratively.

4.5.1 Binarized Back-to-back Bundling as a Hardware-Friendly Approach for Approximate Bundling. We propose a binarized implementation of an approximate bundling operation by reusing the MAN module. It continuously stays in the binary space during the execution of the bundling operation, hence it enables efficient *online and incremental* updates to the prototypes of the AM. The first step is to avoid trying to store the current majority vote and instead bundling the hypervectors iteratively, giving every vote a certain "weight." This is achieved by assigning them a certain chance to be capable of turning the majority around. However, the vote is only turned around if the current one is different.

The first vote has a probability of $P = 1$, the second $P = 1/2$ and so on. Generally the i-th vote has a probability of $P_i = 1/i$ to be able to turn the majority around. Considering all dimensions of the hypervector, this probability turns into a weight. In an abstract sense, these probabilities can be hardwired into the architecture with a connectivity matrix. For large dimensions, the m-th row shows $\approx D/m$ connections, which determine whether the vote at that position can turn around the majority. The maximum number of hypervectors in the bundling record (i.e., the rows in the connectivity matrix) should be predetermined. Figure 15 shows an example of connectivity matrix to bundle 10 hypervectors with dimensionality $D = 64$.

We refer to the example of bundling three hypervectors, where two are equal and one is orthogonal. When bundling with the proposed approach, the orthogonal hypervector is not lost, but is similar to the record as shown in Figure 16 (c.f. Figure 12). Furthermore, when interchanging this approximate method with the ordinary majority vote, the classification accuracy does not change.

As suggested, these characteristics can be implemented using the MAN module to generate a hypervector which is similar to the current bundled hypervector, where the Hamming distance (i.e.,



(a) $a \perp b \perp c$ (b) $a = b \perp c$

Fig. 16. Similarity between three hypervectors and the bundled hypervector $a \oplus b \oplus c$ using binarized back-to-back bundling.

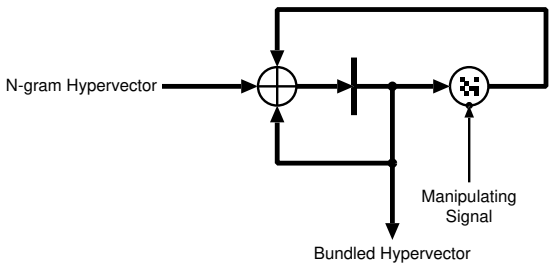


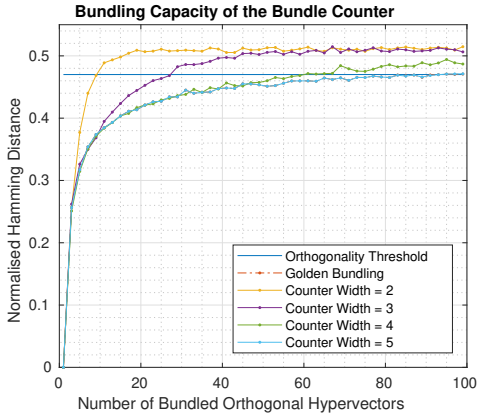
Fig. 17. Data dependency graphs of the binarized back-to-back bundling to approximate temporal majority gate.

the degree of similarity) is determined by the connectivity matrix. Then, the majority vote of three hypervectors is calculated from the input N-gram hypervector, the current bundled hypervector, and it's derived similar (manipulated) hypervector as depicted in Figure 17. The similar hypervector gives the input N-gram hypervector a weight of $1/i$ and the current bundled hypervector a weight of $1 - 1/i$. Compared to the bundling with saturating counters, this approach is far more efficient since it only requires a memory of D bits (fully binarized) without adders and saturation logic.

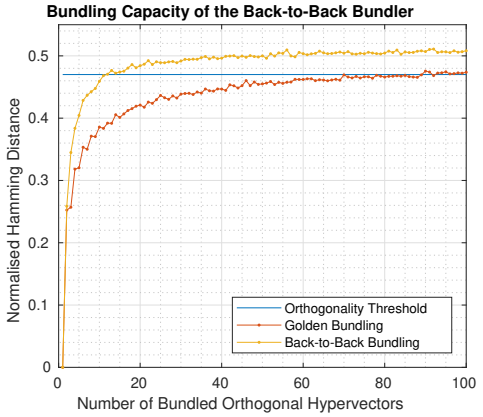
4.5.2 Hypervector Capacity of different Bundling Approaches. The proposed approximate bundling method slightly decreases the capacity of hypervectors. Although for similar hypervectors, as it is the case for N-gram hypervectors among a class (opposed to the bound hypervectors in the spatial encoder), a large capacity is not a requirement. Nevertheless, it is necessary to evaluate how much information a hypervector can store, or how many hypervectors can be bundled into a hypervector (i.e., the capacity of a bundling method).

The capacity can be measured by bundling an increasing number of orthogonal hypervectors and trying to recall the information by measuring the similarity between the bundled hypervector and all compound hypervectors. As long as none of the compound hypervectors crosses the orthogonality threshold (see Section 2.1), their information is still contained in the bundled hypervector. As soon as one of the compound hypervectors becomes orthogonal to the bundled, the bundling method has failed to capture all the information.

For comparison, the ordinary majority vote (see Section 2.2) is used as the reference bundling method. This approach is referred to as the *golden* method. The two other approaches are the



(a) Capacity of the bundle counter (BC) with different widths.



(b) Capacity of back-to-back (B2B) bundling.

Fig. 18. Capacity of different bundling approaches compared with the golden method for a dimensionality of $D = 10000$. The graphs show the maximum normalized Hamming distance between the bundled hypervector and its compound hypervectors.

binarized *back-to-back (B2B)* method from Section 4.5.1 and the *bundle counter (BC)* method (Section 4.2.2), which can be viewed as a very close approximation of the golden method.

The capacity of the binarized back-to-back method in comparison with the golden method is depicted in Figure 18b. The golden method is capable of storing the information of about 60–70 orthogonal hypervectors for a dimensionality of $D = 10000$, whereas the back-to-back binary method saturates between 10–15 hypervectors.

However, the capacity of the counter method is dependent on the number of bits (i.e., width) used to represent the current vote. The smaller the width, the fewer the resources required but the smaller its capacity. This can be seen in Figure 18a. We observe that a width of 5 bits is sufficient to achieve the same capacity as the golden method. When bundling fewer hypervectors, the width should be adjusted to ones needs to minimize the required resources.

4.6 Library: Temporal Encoder Modules

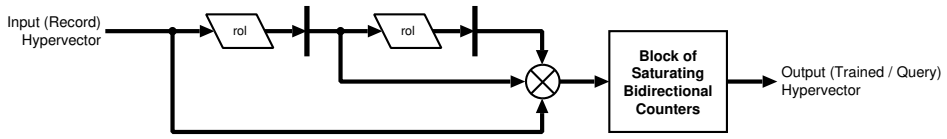
The following library modules emerged from the optimizations in Section 4.4 and 4.5:

- *BC.* A temporal encoder architecture using counter-based bundling as described in Section 4.4 and 4.2.2. See Figure 19a.
- *B2B.* A temporal encoder architecture using manipulator-based back-to-back binary bundling as described in Section 4.4 and 4.5.1. See Figure 19b.

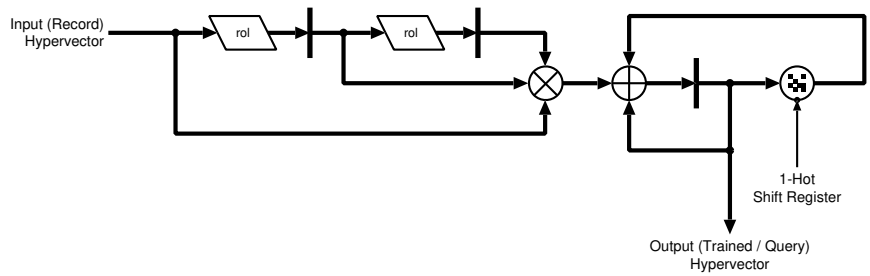
4.7 Associative Memory (AM)

The associative memory (AM) is the part of the architecture that is the most challenging to optimize. One reason is the memory required to store the "trained", prototype, or rather the bundled hypervectors that represent the classes. Another reason is the nature of the Hamming distance, that has to be computed between the query hypervector—of which we want to find the class it belongs to—and each trained hypervectors.

As described in Section 2.1, the Hamming distance measures the number of positions at which two hypervectors differ. This is equal to computing the population count of a hypervector binding



(a) Bundle counter (BC): Using saturating bidirectional counters to bundle N-gram hypervectors.



(b) Binarized Back-to-back (B2B): Using the MAN module to approximate bundling of N-gram hypervectors.

Fig. 19. The temporal encoder architectures available in the library.

those two hypervectors. So far, digital methods for AMs count through all components resulting in a classification latency in the order $O(D)$ [8, 9, 29, 32]. We focus on reducing this latency by adding up all hypervector components.

4.7.1 Deep Adder Trees. When trying to add up all bits of a hypervector, working with tree structures is the most efficient way. In this manner, the AM takes only one clock cycle to compute the Hamming distance, at a cost to long logic delay and gate counts. For a perfect binary tree, which is the case for hypervectors of dimension $D = 2^n$, the depth is $\log_2(D) = n$ which is also the number of adder stages. The amount of adders in stage i is $D/2^i$ and the width of the adders in stage i equals to i. In the simple case of using ripple-carry-adders, the logic delay of the adder tree is equal to $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ delays of a 1-bit-adder. For a dimension $D = 2^{13} = 8192$, this amounts to the delay of 91 1-bit-adders, which will most likely result in the longest path in the architecture. This could be reduced with pipeline registers close to the root, i.e., the final result. The total equivalent of 1-bit-adders for the whole tree can be calculated as follows: $\sum_{i=1}^{n} \frac{D \cdot i}{2^i}$ which, for a dimension $D = 2^{13}$ yields 16369 1-bit-adders.

Although this number of adders seems very high, an FPGA can handle it easily with lookup tables. Furthermore, using the counters as an alternative might seem resource friendlier at first, but turns out an incompetent choice. The reason is that each bit of the hypervector somehow has to be directed to the counter. This requires either huge multiplexers or shift registers with input multiplexers, which both leads to immense area overhead. While the overhead is considerable, the cycles needed to compute the Hamming distance is of the order $O(D)$. This is a poor trade-off compared to the high throughput and moderate overhead of adder tree architectures.

Using the adder trees to compute the Hamming distance between two hypervectors, two AM variations emerge. A fully parallel architecture with replicated adders, leading to $O(1)$ computation cycles, and a vector-sequential architecture, which shares one adder tree to compute the Hamming distance of all hypervectors one after the other, hence leading to $O(n_{\text{classes}})$ computation cycles.

Table 2. Parameter configuration for the case study.

Parameter	Value
Hypervector Dimension (D)	8192
Channels	4
Classes	5
Quantization	21
N-gram Size	3
Bundle Counter Width (*MAN & CA*)	3
Bundle Counter Width (*BC*)	5
Max. Bundle Cycles (*B2B*)	256

4.8 Library: Associative Memory Modules

The following library modules emerged from the optimizations in Section 4.7:

- *BS*. A bit-sequential AM architecture. This is the starting point for optimizations and was described in [8, 9, 29, 32]. See Figure 20a.
- *CMB*. An AM architecture based on adder trees as described in Section 4.7.1. See Figure 20b.
- *VS*. A vector-sequential AM architecture based on adder trees as described in Section 4.7.1. See Figure 20c.

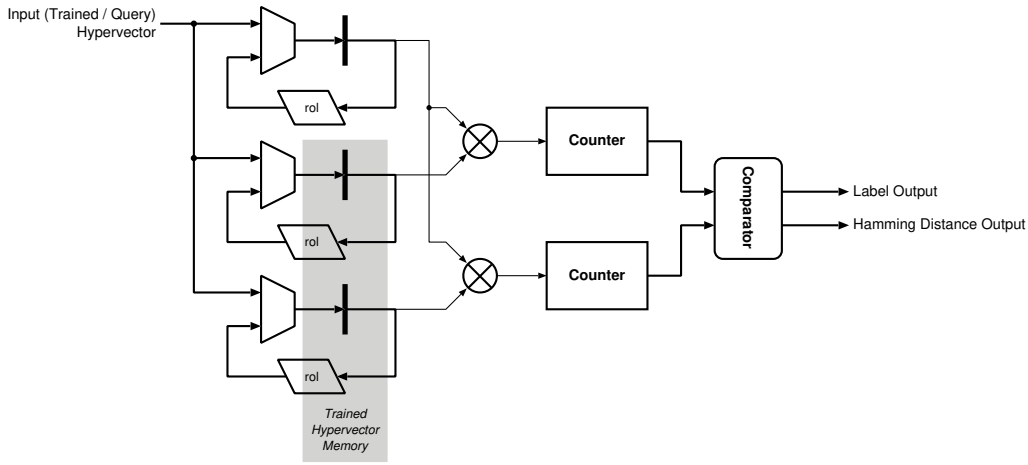
5 DESIGN SPACE EXPLORATION AND EXPERIMENTAL RESULTS

In order to evaluate the library modules, they are configured for the EMG-based hand gesture recognition task, and all possible combinations of HD architectures (i.e., our design space) are synthesized for a Xilinx® Virtex UltraScale™ FPGA [41]. All the HD architectures are functionally equivalent and exhibit iso-accuracy. The parameters for the configured architectures are listed in Table 2. The library can be configured to conduct virtually any learning and classification task.

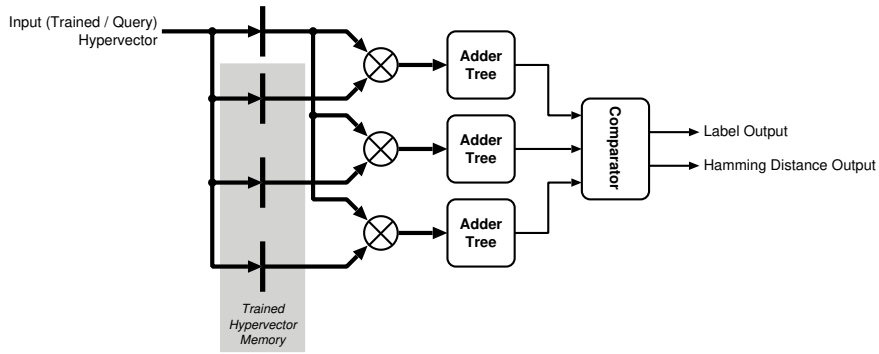
Each HD architecture is composed of three modules in series: a type of mapping and spatial encoder followed by a type of temporal encoder, and finally a type of AM. To conduct the design space exploration, each architecture's throughput is plotted against its area efficiency (defined as 1/CLBs) in Figure 21. The quality of an architecture increases when going from left to right and/or bottom to top. The color coding represents HD architectures with the same type of AM.

Our starting point is the LUT+BC+BS architecture as an improved version of [28] using bidirectional saturating counters. What can be observed is that by replacing the LUT module with the proposed MAN and CA modules, a significant area saving is achieved. This area saving is consistent with any combination of temporal encoder and AM. A similar area improvement can be observed when replacing the BC module with the novel B2B module. Combining both optimization leads to an area improvement of up to ×2.39. On the other hand, a massive throughput improvement of up to ×2337 can be achieved by moving from an AM with the BS module to VS and finally CMB.

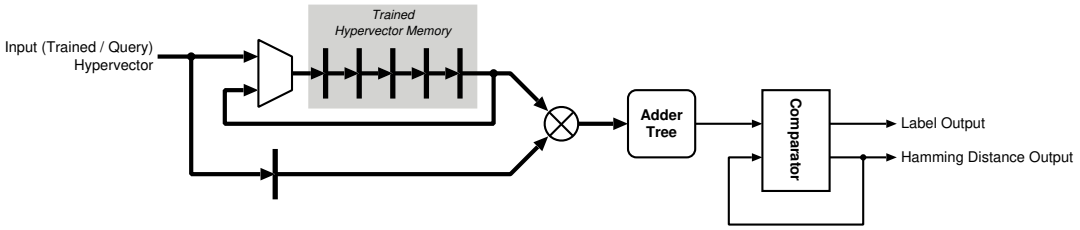
Different combinations of the modules produce architectures with varying area/throughput improvements. Eventually, four architectures stand out as pareto optimal architectures (see Table 3). These offer different trade-offs and can be selected depending on the user's requirements. The throughput of these architectures is significantly higher than the classification constraint for real-time EMG tasks [2, 17]. Note that different configurations may lead to different pareto optimal architectures.



(a) Bit-sequential (BS): This AM has a latency of $O(D)$.



(b) Combinational (CMB): This single-cycle AM has a latency of $O(1)$.



(c) Vector-sequential (VS): This AM has a latency of $O(n_{\text{classes}})$.

Fig. 20. The associative memory architectures available in the library.

5.1 Scalability: Larger number of Channels and Classes

Here, we assess the scalability of our proposed methods when doubling the number of channels and classes. The spatial encoder with the CA module shows the best area efficiency for applications with a larger number of channels, followed by the spatial encoder with the MAN module. The memory

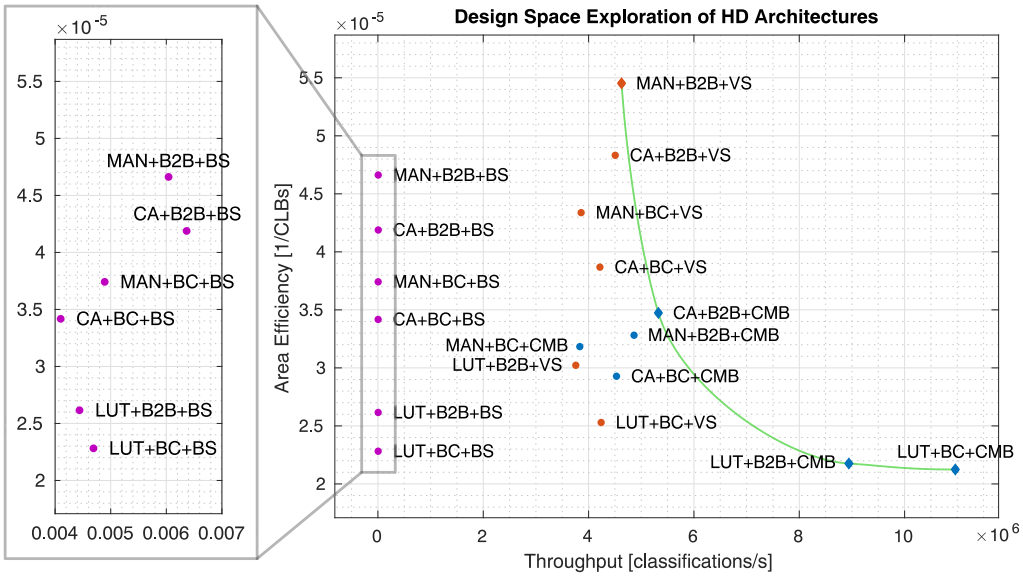


Fig. 21. Design space exploration of HD architectures using all possible combinations of the modules available in the library. Colors indicate the architectures with the same type of AM. Pareto optimal architectures are marked with a diamond ◆ and connected by a green line representing the Pareto frontier.

Table 3. Area and throughput results of the "starting point" and the Pareto optimal architectures.

Architecture	**Throughput** [classifications/s]	**Throughput Improvement**	**Area** [CLBs]	**Area Improvement**
LUT+BC+BS	$4.69 \cdot 10^3$	×1	43825	×1
MAN+B2B+VS	$4.62 \cdot 10^6$	×986	18340	×2.39
CA+B2B+CMB	$5.33 \cdot 10^6$	×1136	28788	×1.52
LUT+B2B+CMB	$8.94 \cdot 10^6$	×1906	45961	×0.95
LUT+BC+CMB	$10.96 \cdot 10^6$	×2337	47068	×0.93

footprint of CA module is independent of the number of channels since only a seed hypervector to initialize the CA state needs to be stored, hence the area will not increase (see Table 4a). However, it requires almost twice clock cycles to produce the channel hypervectors for the doubled number of channels. The spatial encoder with the LUT shows opposite scalability: it maintains almost the same throughput but increases the area by 2.41×. Focusing on the AM module, an application with twice the number of classes will impose a larger area to the CMB and BS modules, whereas the VS' area is mostly unaffected, apart from the storage for additional trained hypervectors (see Table 4b).

6 CONCLUSIONS

This paper proposes hardware optimizations—in an open-source VHDL library—for dense binary HD computing that enable efficient synthesis of acceleration engines handling both inference and training tasks on an FPGA. The Pareto optimal design is mapped on only 18340 CLBs of a Xilinx® UltraScale™ FPGA achieving simultaneous 2.39× lower area and 986× higher throughput compared

Table 4. Scalability of the library modules.

(a) Throughput and area scaling of the spatial encoder modules when doubling the number of channels from 4 to 8.

Module	Throughput Scaling	Area Scaling
LUT	×0.94	×2.41
CA	×0.45	×0.99
MAN	×0.61	×1.01

(b) Throughput and area scaling of the AM modules when doubling the number of classes from 6 to 12.

Module	Throughput Scaling	Area Scaling
BS	×0.49	×1.89
CMB	×0.63	×2.14
VS	×0.59	×1.10

to the baseline. This is accomplished by: (1) rematerializing hypervectors on the fly by substituting the cheap logical operations for the expensive memory accesses to seed hypervectors; (2) online and incremental learning from different gesture examples while staying in the binary space; (3) combinational associative memories to steadily reduce the latency of classification. Our future work will target an ASIC implementation of the library modules.

ACKNOWLEDGMENTS

Support was received from the ETH Zurich Postdoctoral Fellowship program, the Marie Curie Actions for People COFUND Program, and the European Union's Horizon 2020 Research and Innovation Program through the project MNEMOSENE under Grant 780215.

REFERENCES

[1] Alessio Burrello, Kaspar Schindler, Luca Benini, and Abbas Rahimi. 2018. One-shot learning for iEEG seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing. In *Biomedical Circuits and Systems Conference (BioCAS), 2018 IEEE*.

[2] J. U. Chu, I. Moon, and M. S. Mun. 2006. A Real-Time EMG Pattern Recognition System Based on Linear-Nonlinear Feature Projection for a Multifunction Myoelectric Hand. *IEEE Transactions on Biomedical Engineering* 53, 11 (Nov 2006), 2232–2239. https://doi.org/10.1109/TBME.2006.883695

[3] Chris Eliasmith. 2013. *How to Build a Brain: A Neural Architecture for Biological Cognition.* Oxford Series on Cognitive Models and Architectures.

[4] B. Emruli, R. W. Gayler, and F. Sandin. 2013. Analogical mapping and inference with binary spatter codes and sparse distributed memory. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN.2013.6706829

[5] Ross W. Gayler. 1998. Multiplicative Binding, Representation Operators & Analogy. In *Gentner, D., Holyoak, K. J., Kokinov, B. N. (Eds.), Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences.* New Bulgarian University, Sofia, Bulgaria, 1–4. http://cogprints.org/502/

[6] Ross W. Gayler. 2003. Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience. In *Proceedings of the Joint International Conference on Cognitive Science. ICCS/ASCS.* 133–138.

[7] P. C. Huang and J. M. Rabaey. 2017. A Bio-Inspired Analog Gas Sensing Front End. *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, 9 (Sept 2017), 2611–2623. https://doi.org/10.1109/TCSI.2017.2697945

[8] M. Imani, D. Kong, A. Rahimi, and T. Rosing. 2017. VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*. 1–8. https://doi.org/10.1109/ICRC.2017.8123650

[9] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey. 2017. Exploring Hyperdimensional Associative Memory. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 445–456. https://doi.org/10.1109/HPCA.2017.28

[10] Aditya Joshi, Johan T. Halseth, and Pentti Kanerva. 2017. Language Geometry Using Random Indexing. In *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20-22, 2016, Revised Selected Papers*, Jose Acacio de Barros, Bob Coecke, and Emmanuel Pothos (Eds.). Springer International Publishing, Cham, 265–274. https://doi.org/10.1007/978-3-319-52289-0_21

[11] Pentti Kanerva. 1988. *Sparse Distributed Memory.* MIT Press Cambridge.
[12] Pentti Kanerva. 1996. Binary Spatter-Coding of ordered k-tuples. In *ICANN'96, Proceedings of the International Conference on Artificial Neural Networks (Lecture Notes in Computer Science)*, (Ed.), Vol. 1112. Springer, 869–873.
[13] Pentti Kanerva. 2009. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation* 1, 2 (2009), 139–159. https://doi.org/10.1007/s12559-009-9009-8
[14] Pentti Kanerva. 2010. What We Mean When We Say "What's the Dollar of Mexico?": Prototypes and Mapping in Concept Space. In *AAAI Fall Symposium: Quantum Informatics for Cognitive, Social, and Semantic Processes*. 2–6.
[15] Pentti Kanerva. 2014. Computing with 10,000-Bit Words. In *Proc. 52nd Annual Allerton Conference on Communication, Control, and Computing*.
[16] Pentti Kanerva, Jan Kristoferson, and Anders Holst. 2000. Random Indexing of Text Samples for Latent Semantic Analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*. Erlbaum, 1036. http://www.rni.org/kanerva/cogsci2k-poster.txt
[17] Mahdi Khezri and Mehran Jahed. 2007. Real-time intelligent pattern recognition algorithm for surface EMG signals. *BioMedical Engineering OnLine* 6, 1 (03 Dec 2007), 45. https://doi.org/10.1186/1475-925X-6-45
[18] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey. 2018. Classification and Recall With Binary Hyperdimensional Computing: Tradeoffs in Choice of Density and Mapping Characteristics. *IEEE Transactions on Neural Networks and Learning Systems* (2018), 1–19. https://doi.org/10.1109/TNNLS.2018.2814400
[19] Simon D. Levy, Suraj Bajracharya, and Ross W. Gayler. 2013. Learning Behavior Hierarchies via High-dimensional Sensor Projection. In *Proceedings of the 12th AAAI Conference on Learning Rich Representations from Low-Level Sensors (AAAIWS'13-12)*. AAAI Press, 25–27. http://dl.acm.org/citation.cfm?id=2908225.2908230
[20] H. Li, T. F. Wu, A. Rahimi, K. S. Li, M. Rusch, C. H. Lin, J. L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, W. C. Chiu, M. C. Chen, T. T. Wu, J. M. Shieh, W. K. Yeh, J. M. Rabaey, S. Mitra, and H. S. P. Wong. 2016. Hyperdimensional computing with 3D VRRAM in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition. In *2016 IEEE International Electron Devices Meeting (IEDM)*. 16.1.1–16.1.4. https://doi.org/10.1109/IEDM.2016.7838428
[21] A. Moin, A. Zhou, A. Rahimi, S. Benatti, A. Menon, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, L. Benini, A. C. Arias, and J. M. Rabaey. 2018. An EMG Gesture Recognition System with Flexible High-Density Sensors and Brain-Inspired High-Dimensional Classifier. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS.2018.8351613
[22] Fabio Montagna, Abbas Rahimi, Simone Benatti, Davide Rossi, and Luca Benini. 2018. PULP-HD: Accelerating Brain-inspired High-dimensional Computing on a Parallel Ultra-low Power Platform. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*. ACM, New York, NY, USA, Article 111, 6 pages. https://doi.org/10.1145/3195970.3196096
[23] Fateme Rasti Najafabadi, Abbas Rahimi, Pentti Kanerva, and Jan M. Rabaey. 2016. Hyperdimensional Computing for Text Classification. *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth* (2016). https://www.date-conference.com/system/files/file/date16/ubooth/37923.pdf
[24] P. Neubert, S. Schubert, and P. Protzel. 2016. Learning Vector Symbolic Architectures for Reactive Robot Behaviours. In *Proc. of Intl. Conf. on Intelligent Robots and Systems (IROS) Workshop on Machine Learning Methods for High-Level Cognitive Capabilities in Robotics*.
[25] T.A. Plate. 1995. Holographic reduced representations. *IEEE Transactions on Neural Networks* 6, 3 (1995), 623–641.
[26] T.A. Plate. 2003. *Holographic Reduced Representations.* CLSI Publications. 300 pages.
[27] D. A. Rachkovskij. 2017. Binary Vectors for Fast Distance and Similarity Estimation. *Cybernetics and Systems Analysis* 53, 1 (01 Jan 2017), 138–156. https://doi.org/10.1007/s10559-017-9914-x
[28] Abbas Rahimi, Simone Benatti, Pentti Kanerva, Luca Benini, and Jan M. Rabaey. 2016. Hyperdimensional Biosignal Processing: A Case Study for EMG-based Hand Gesture Recognition. In *IEEE International Conference on Rebooting Computing*.
[29] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey. 2017. High-Dimensional Computing as a Nanoscalable Paradigm. *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, 9 (Sept 2017), 2508–2521. https://doi.org/10.1109/TCSI.2017.2705051
[30] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey. 2018. Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals. *Proc. IEEE* (2018), 1–21. https://doi.org/10.1109/JPROC.2018.2871163
[31] Abbas Rahimi, Pentti Kanerva, José del R Millán, and Jan M. Rabaey. 2017. Hyperdimensional Computing for Noninvasive Brain–Computer Interfaces: Blind and One-Shot Classification of EEG Error-Related Potentials. *10th ACM/EAI International Conference on Bio-inspired Information and Communications Technologies (BICT)* (2017).
[32] Abbas Rahimi, Pentti Kanerva, and Jan M. Rabaey. 2016. A Robust and Energy Efficient Classifier Using Brain-Inspired Hyperdimensional Computing. In *Low Power Electronics and Design (ISLPED), 2016 IEEE/ACM International Symposium*

on.

[33] Abbas Rahimi, Artiom Tchouprina, Pentti Kanerva, José del R. Millán, and Jan M. Rabaey. 2017. Hyperdimensional Computing for Blind and One-Shot Classification of EEG Error-Related Potentials. *Mobile Networks and Applications* (03 Oct 2017). https://doi.org/10.1007/s11036-017-0942-6

[34] O. Räsänen. 2015. Generating Hyperdimensional Distributed Representations from Continuous Valued Multivariate Sensory Input. In *Proceedings of the 37th Annual Meeting of the Cognitive Science Society*. 1943–1948.

[35] O. Räsänen and S. Kakouros. 2014. Modeling Dependencies in Multiple Parallel Data Streams with Hyperdimensional Computing. *IEEE Signal Processing Letters* 21, 7 (July 2014), 899–903. https://doi.org/10.1109/LSP.2014.2320573

[36] O. Räsänen and J. Saarinen. 2015. Sequence Prediction With Sparse Distributed Hyperdimensional Coding Applied to the Analysis of Mobile Phone Use Patterns. *IEEE Transactions on Neural Networks and Learning Systems* PP, 99 (2015), 1–12. https://doi.org/10.1109/TNNLS.2015.2462721

[37] R. Santoro, S. Roy, and O. Sentieys. 2007. Search for Optimal Five-Neighbor FPGA-Based Cellular Automata Random Number Generators. In *2007 International Symposium on Signals, Systems and Electronics*. 343–346. https://doi.org/10.1109/ISSSE.2007.4294483

[38] Stephen Wolfram. 1986. Random sequence generation by cellular automata. *Advances in Applied Mathematics* 7, 2 (1986), 123 – 169. https://doi.org/10.1016/0196-8858(86)90028-X

[39] T. F. Wu, H. Li, P. Huang, A. Rahimi, G. Hills, B. Hodson, W. Hwang, J. M. Rabaey, H. . P. Wong, M. M. Shulaker, and S. Mitra. 2018. Hyperdimensional Computing Exploiting Carbon Nanotube FETs, Resistive RAM, and Their Monolithic 3D Integration. *IEEE Journal of Solid-State Circuits* 53, 11 (Nov 2018), 3183–3196. https://doi.org/10.1109/JSSC.2018.2870560

[40] T. F. Wu, H. Li, P. C. Huang, A. Rahimi, J. M. Rabaey, H. S. P. Wong, M. M. Shulaker, and S. Mitra. 2018. Brain-inspired computing exploiting carbon nanotube FETs and resistive RAM: Hyperdimensional computing case study. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 492–494. https://doi.org/10.1109/ISSCC.2018.8310399

[41] Xilinx. 2017. UltraScale Architecture and Product Data Sheet: Overview. https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf DS890 (v3.1) November 15, 2017.