Mobile Netw Appl
https://doi.org/10.1007/s11036-017-0942-6

Hyperdimensional Computing for Blind and One-Shot Classification of EEG Error-Related Potentials

Abbas Rahimi[1] · Artiom Tchouprina[2,3] · Pentti Kanerva[4] · José del R. Millán[5] · Jan M. Rabaey[1]

Abstract The mathematical properties of high-dimensional (HD) spaces show remarkable agreement with behaviors controlled by the brain. Computing with HD vectors, referred to as "hypervectors," is a brain-inspired alternative to computing with numbers. HD computing is characterized by generality, scalability, robustness, and fast learning, making it a prime candidate for utilization in application domains such as brain–computer interfaces. We describe the use of HD computing to classify electroencephalography (EEG) error-related potentials for noninvasive brain–computer interfaces. Our algorithm *naturally* encodes neural activity recorded from 64 EEG electrodes to a single temporal–spatial hypervector without requiring any electrode selection process. This hypervector represents the event of interest, can be analyzed to identify the most discriminative electrodes, and is used for recognition of the subject's intentions. Using the full set of training trials, HD computing achieves on average 5% higher single-trial classification accuracy compared to a conventional machine learning method on this task (74.5% vs. 69.5%) and offers further advantages: (1) Our algorithm learns fast: using only 34% of training trials it achieves an average accuracy of 70.5%, surpassing the conventional method. (2) Conventional method requires prior *domain expert* knowledge, or a separate process, to carefully select a subset of electrodes for a subsequent preprocessor and classifier, whereas our algorithm blindly uses all 64 electrodes, tolerates noises in data, and the resulting hypervector is intrinsically clustered into HD space; in addition, most preprocessing of the electrode signal can be eliminated while maintaining an average accuracy of 71.7%.

Keywords Electroencephalogram (EEG) · Error-related potentials (ERP) · Classification · Hyperdimensional computing

1 Introduction

Over the past six decades, the semiconductor industry has been immensely successful thanks to the set of well-defined abstraction layers, starting from robust switching devices that support a deterministic Boolean algebra and going up

✉ Abbas Rahimi
abbas@eecs.berkeley.edu

Artiom Tchouprina
artiom.tchouprina@st.com

Pentti Kanerva
pkanerva@berkeley.edu

José del R. Millán
jose.millan@epfl.ch

Jan M. Rabaey
jan@eecs.berkeley.edu

[1] Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA 94720, USA

[2] STMicroelectronics, Crolles, France

[3] Berkeley Wireless Research Center, Berkeley, CA 94704, USA

[4] Redwood Center for Theoretical Neuroscience, University of California Berkeley, Berkeley, CA 94720, USA

[5] Defitech Foundation Chair in Brain-Machine Interface, Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland

Published online: 03 October 2017

to a scalable and stored program architecture, which is Turing complete, and hence capable of tackling (almost) any computational task. Unfortunately this abstraction chain is being challenged as device scaling continues to nanometer dimensions and also by exciting new applications that must support a myriad inputs such as brain–computer interfaces [1].

A brain–computer interface is a device that enables communication and control without movement. For these applications, cognitive functions such as classification, recognition, synthesis, decision-making and learning are of crucial importance for fast and 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. It is therefore worth exploring alternative computational models for emerging applications by abandoning the deterministic requirement.

Brain-inspired information processing architectures provide significant increase in energy efficiency, asymptotically approaching the efficiency of brain computation, while aligning well with the variability of nanoscale devices [10, 17, 20, 23]. The mathematical properties of high-dimensional spaces correlate strongly with behaviors controlled by the brain [13, 15, 16, 24]. We focus on a model of computing with high-dimensional (HD) vectors—hereafter HD computing—where the dimensionality is in the thousands (e.g., 10,000 dimensions). HD computing is also referred to as "hyperdimensional computing" [13] on account of the very high dimensionality. In this formalism, information is represented in HD vectors referred to as "hypervectors". HD computing explores the emulation of cognition by computing with hypervectors as an alternative to computing with bits and numbers.

Key properties of HD computing include: (1) HD computing paradigm is universal and complete. (2) By its very nature, HD computing overcomes large variability and uncertainty in both data and implementation platform to perform robust decision making and classification. (3) In contrast to other neuro-inspired approaches, in which learning is separate from subsequent execution, learning in HD computing shares its constructs with execution, is relatively fast, and does not rely on biologically unlikely algorithms such as back-propagation. (4) In HD computing, the encoding of information and memory storage are separate processes. In contrast, in the artificial neural nets these are entangled which can lead to catastrophic forgetting. Such generality, robustness against data uncertainty, and one-shot learning make HD computing a prime candidate for utilization in application domains such as brain–computer interfaces and wearable cyberbiological systems.

HD computing has been used for text analytics solely from a stream of input letters. More specifically, HD computing can identify the language of unknown sentences from 21 European languages [11, 23], and classify Reuters news articles to eight topics [19] with very high accuracy. It has also been adapted to wearable biosignal processing characterized by a set of parallel and analog streaming inputs. Very simple vector-space operations are used to classify hand gestures from four electromyography (EMG) electrodes [21].

We further extend the application domain of HD computing to noninvasive brain–computer interfaces in [22], and in this paper explain it with more details. We develop an encoding algorithm and a classifier[1] for recognition of the subject's intentions from error-related electroencephalography (EEG) potentials. Our algorithm encodes neural activity that is recorded simultaneously by 64 EEG electrodes, to a single temporal–spatial hypervector representing the intention of a subject. Our proposed HD classifier surpasses the state-of-the-art method [4]—referred to as baseline in this paper—for classifying single-trial EEG error-related potentials in the following aspects:

1. **Higher accuracy**. With an equivalent setup, HD classifier achieves an average classification accuracy of 74.5% which is 5% higher than the baseline.
2. **Faster learning**. HD classifier learns $\approx 3\times$ faster by using only 34% of training trials while maintaining an average accuracy of 70.5% which is higher than the baseline using the full set of training trials.
3. **No electrode selection**. A particularly important aspect of our HD classifier is that it can maintain high accuracy while blindly using all 64 electrodes. Authors in [4] carefully selected one or two electrodes, depending upon the subject, to be used for the baseline classifier. In this paper, we also observe that in fact those selected electrodes have a high Fisher score for discrimination of the subject's intentions. Nevertheless, our HD classifier does not require such *domain expert* knowledge or electrode selection process, and hence operates *naturally* with all 64 electrodes at negligible loss of accuracy.
4. **Analyzable code.** The learned code (i.e., the temporal–spatial hypervector) can be analyzed to extract useful information. For example, the most discriminating electrodes can also be identified by measuring the relative distances between the learned class prototypes in the HD space. This feature enables standard verification of

[1] MATLAB code for our encoding algorithm and classifier is open access and available at https://github.com/abbas-rahimi/HDC-EEG-ERP.

the learned model and is in sharp contrast to conventional learning methods that produce a “black box.”

5. **Less preprocessing**. Most preprocessing of the electrode signal can be eliminated in our classifier. Such “blind” operation with all electrodes and less preprocessing results in only a slight loss of accuracy (from 74.5 to 71.7%).

This paper is organized as follows. In Section 2, we describe EEG error-related potentials and its usage in noninvasive brain–computer interfaces followed by its baseline preprocessing and classification. In Section 3, we introduce HD computing and discuss how its operations can be used to form an HD classifier. In Section 4, we present our algorithm for encoding and classifying EEG error-related potentials. In Section 5, we provide experimental results followed by discussion in Section 6. Section 7 concludes this paper.

2 Noninvasive brain–computer interfaces

Noninvasive brain–computer interfaces and neuroprostheses aim to provide a communication and control channel based on the recognition of the subject’s intentions from spatiotemporal neural activity typically recorded by EEG electrodes. What makes it particularly challenging, however, is its susceptibility to errors in the recognition of human intentions.

2.1 EEG error-related potentials

As an alternative interaction approach, the user can observe the performance of an autonomous agent endowed with learning capabilities, and the erroneous behavior of the agent can be recognized directly from the analysis of the user’s brain signals, i.e., EEG error-related potentials (ERP). In the frame of brain–computer interaction, Ferrez and Millán have described an ERP elicited by errors in the recognition of the user’s intention when operating a brain–computer interface [6, 7]. In their experimental protocol, the human subject tries to move a cursor toward a target location either using a keyboard [6] or mental commands [7]. Next, they have observed that similar error-related signals are generated when a human user observes the performance of an external agent upon which the user has no control [4]. In this approach, the user does not provide any commands, but only observes the agent’s performance. Efficient and fast learning methods of encoding these ERPs for accurately classifying the user’s intentions with regard to the agent further motivates its application for noninvasive brain–computer interfaces.

2.2 Dataset for error-related potentials

Here, we first describe a publicly available dataset [2] for ERPs, and then outline the baseline method [4] that is used for processing of these potentials. Six subjects are seated in front of a computer screen where a moving cursor is displayed. A colored square at either the left or right of the cursor indicates a target location. At each trial the cursor moves toward or away from the target. During the experiment, shown in Fig. 1, the user has no control over the cursor’s movement and is asked only to observe the performance of the agent that controls the cursor, knowing that the goal is to reach the target. To study signals generated by erroneous actions, at each trial, there is a probability of ≈0.20 for the cursor to move in the wrong direction (i.e., opposite to the target location). A trial is labeled as “correct” if the curser is moved toward the target; otherwise it is labeled as “error”, e.g., when the target is located in the right and the cursor moves to the left, as in Fig. 1.

Trials have an approximate duration of 2000 ms. There are two recording sessions, the first one is used for training, and the second one is used for testing. Each experimental session consists of 10 blocks of 3 min each (≈ 64 trials per block). Full details of the experimental protocol are provided in [4]. In the following, we explain their methods for EEG signal acquisition, preprocessing, and classification. We refer to them as the baseline for comparing with our HD computing method.

2.2.1 Baseline preprocessing and classification

EEG potentials were recorded at a sampling rate of 512 Hz using 64 electrodes according to the standard 10/20 international system. For preprocessing, data was spatially filtered using common average reference (CAR) [18]. By applying the CAR filter to an electrode, the average signal level of the entire electrode array is subtracted from that of the

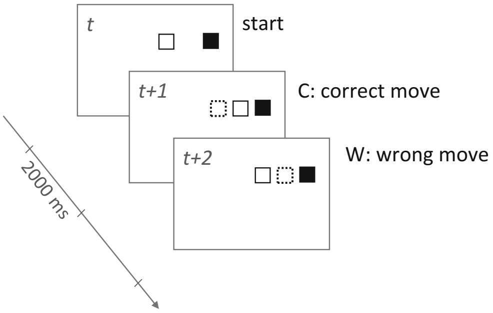


Fig. 1 Experimental Protocol of ERPs: black square as target location; white square as moving cursor; and dotted square as cursor location at the previous time step

electrode of interest. If the entire head is covered by equally spaced electrodes and the potential on the head is generated by point sources, the CAR results in a spatial voltage distribution with a mean of zero [3]. We show in Section 5.3 that this spatial filter preprocessing can be eliminated with negligible effect on the classification accuracy thanks to the HD operations that can work on raw data. Then, a 1–10 Hz band-pass filter (BPF) was applied to remove the unwanted frequency components. For every subject, a time window corresponding to erroneous and correct cursor movements was extracted for further analysis and classification (listed in the third column of Table 1).

A Gaussian classifier was used for recognition of a single trial, as described in [6]. This statistical classifier estimates the posterior probability of a given trial corresponding to one of the two classes: "error," and "correct." FCz and Cz electrodes were used as the inputs to the classifier following their earlier studies for electrode selection process [5]. The same learning rates and number of prototypes were used in all cases. Classifier parameters are then tuned using a stochastic gradient descent on the mean square error [6]. To tune the classification performance, the choice of electrodes (FCz, Cz, or both) and time windows were selected independently per subject [4] (see Table 1).

Our aim is to develop an efficient and fast learning method based on HD computing that replaces the aforementioned preprocessing and classification enabling blindly operating with all electrodes, and with raw data. We provide backgrounds about HD computing in Section 3, and then present details of our method in Section 4.

3 HD computing

The brain's circuits are massive in terms of numbers of neurons and synapses, suggesting that large circuits are fundamental to the brain's computing. HD computing [13, 15] explores this idea by looking at computing with ultra-wide words – that is, with hypervectors. There exist a huge number of different, nearly orthogonal hypervectors with the dimensionality in the thousands (e.g., $D = 10{,}000$) [12]. This lets us combine two such hypervectors into a new hypervector using well-defined vector-space operations, while keeping the information of the two with high probability.

Table 1 Classifier parameters: selected electrodes and time windows used in Gaussian classifier [4]; and n-gram sizes for our HD classifier

Subjects	Electrodes	Time window (ms)	n-gram
S1	FCz, Cz	200–450	16
S2	Cz	150–600	29
S3	FCz, Cz	200–450	16
S4	FCz	0–600	19
S5	FCz, Cz	150–600	29
S6	FCz, Cz	150–600	29

Computing with hypervectors begins with selecting a set of random HD vectors to represent basic objects. These vectors are also thought of as *labels*. In a language application, for example, letters of the alphabet can be the basic objects and they are assigned random labels, and in dealing with EEG signals each channel is assigned a random label, independently of all the other labels. They serve as *seed vectors*, and they are used to make representations for more complex objects. For seeds, we use random 10,000-dimensional vectors of equally probable 1s and −1s, i.e., dense bipolar elements of $\{1, -1\}^{10000}$.

3.1 Item memory

Item memory (iM) is a symbol table or dictionary of all the hypervectors defined in the system. In a typical language application, the 26 letters of the alphabet and the space are the initial items, and they are assigned hypervectors at random (with i.i.d. components). They stay fixed throughout the computation, and they serve as seeds from which further representations are made. HD computing has been used for identifying the source language of text samples from a sequence of n consecutive letters (n-grams) [11, 23]. For example, letter trigrams of a text sample were encoded into a hypervector by random indexing and vector-space operations to represent a language. In the same vein, pentagrams of letters have been used for classifying news articles [19].

Text and language applications are well-matched to the HD computing framework because the input data already comes in the form of discrete symbolic primitives (letters, or words), which can be readily mapped to hypervectors. On the other hand, biosignal processing applications often operate on analog time series with multiple sensory inputs demanding a different mapping scheme to hypervectors. Accordingly, we have extended the notion of iM to a *continuous* item memory (CiM) that maps an analog input after a discretization step [21]. CiM utilizes a method of mapping quantities and dates "continuously" to hypervectors [25]. In this continuous vector space, orthogonal endpoint hypervectors are generated for the minimum and maximum levels in the range. Hypervectors for intermediate levels are then generated by linear interpolation between these endpoints so that the similarity of vectors corresponds to the closeness of levels.

For example, if an analog signal is discretized into m levels, we choose a random hypervector for the minimum level and randomly flip $D/2/(m-1)$ of its bits for each successively higher level (once flipped, a bit will not be flipped back). The vectors for the minimum and the maximum levels will then be $D/2$ bits apart or orthogonal to each other. These hypervectors are stored in the CiM and used to encode signal levels.

3.2 MAP operations

The seed hypervectors that are stored in iM and CiM can be further combined using the following well-defined set of arithmetic operations. We consider a variant of the multiplication, addition, and permutation (MAP) coding described in [8] to define the hyperdimensional vector space. The MAP operations on the hypervectors are defined as follows. Point-wise multiplication of two hypervectors A and B is denoted by $A * B$, and point-wise addition is denoted by $A + B$. Multiplication takes two vectors and yields a third, $A * B$, that is dissimilar (orthogonal) to the two and is suited for variable binding; and addition takes several vectors and yields their mean vector $[A + B + ... + X]$ that is maximally similar to them and is suited for representing sets. The brackets $[\cdots]$ mean that the sum vector is normalized to $\{+1, -1\}^{10000}$ based on the sign, with ties broken at random. In the following, we describe the use of these two operations to holistically encode a data record composed of various fields [14].

A data record consists of a set of fields (variables/attributes) and their values (fillers); for example, the variables x, y, z with values a, b, c, respectively. The holistic encoding is done as follows. The field–value pair $x = a$ is encoded by the hypervector $X * A$ that binds the corresponding hypervectors, and the entire record is encoded by the hypervector $R = [(X * A) + (Y * B) + (Z * C)]$ which includes both the variables and the values, and each of them spans the entire 10,000-bit hypervector.

Finally, the third operation is a permutation, ρ, that rotates the hypervector coordinates. It is implemented as a cyclic right-shift by one position. The permutation operation generates a dissimilar pseudo-orthogonal hypervector that is good for encoding sequences. In geometry sense, the permutation rotates the hypervector in the space. For example, the sequence trigram *abc* is encoded as the hypervector $\rho(\rho A{*}B){*}C = \rho\rho A{*}\rho B{*}C$. This efficiently distinguishes the sequence *abc* from *acb*, since a rotated hypervector is uncorrelated with all the other hypervectors.

3.3 Associative memory

Hypervectors can be stored in an associative memory to be compared for similarity using a distance metric over the vector space. We use cosine similarity as the distance metric between two hypervectors by measuring the cosine of the angle between them using a dot product. It is defined as $\cos(A, B) = |A' * B'|$, where A' and B' are the length-normalized vectors of A and B, respectively, and $|C|$ denotes the sum of the elements in C. It is thus a measure of orientation and not magnitude: two hypervectors with the same orientation have a cosine similarity of 1, two orthogonal hypervectors have a similarity of 0, and two hypervectors diametrically opposed have a similarity of -1.

4 HD computing for EEG ERP

In this section, we describe how HD computing can be used to encode ERPs into hypervectors. Our proposed encoder first captures a sequence of electrical activities of an electrode into a temporal hypervector (Section 4.1) and then encodes the information across all the electrodes into a temporal–spatial hypervector (Section 4.2) for the HD classifier (Section 4.3). Figure 2 shows the structure of proposed temporal–spatial encoder.

4.1 Temporal encoder for one electrode

There are 64 EEG electrodes with unique names, and each electrode produces an analog signal with an amplitude. We draw an analogy from [14] to generate a holistic vector representing information about a given electrode by using a field–value pair. Hence, we decouple the *name* and the *signal level* of an electrode. The electrode name corresponds to a field of a traditional data record, and its signal level corresponds to the value for the field. Since the name of every electrode is a unique string, it forms a field that can be easily mapped to a hypervector (N) using an iM with 64 entries. The iM represents the 64 basic fields by assigning a unique orthogonal hypervector to every field: $N_1 \perp N_2 \ldots \perp N_{64}$.

$$N_i = \text{iM (name of } i\text{th electrode)} \qquad (1)$$

Although the names of electrodes can be readily mapped to hypervectors, mapping their signal levels requires a quantization step. Here, the signal level can be the raw analog data or the preprocessed data as described in Section 2.2.1. The signal levels of each electrode are scaled linearly from 0 to 100, and quantized into 100 discrete levels. This quantized signal level is mapped to a hypervector (L) using a CiM with 100 entries. In CiM, the hypervector that is assigned to the minimum signal level is orthogonal to the hypervector representing the maximum signal level: CiM(0) $\perp$ CiM(99).

An event of ERP is not a single signal sample but rather a sequence of samples spanned over a time window forming a *temporal* component. Hence, we should design a temporal encoder to capture all the signal levels generated during the entire event of an ERP. We can encode a sequence of symbols by using the permutation operation, ρ. As shown in Section 3.2, permutation has been used to

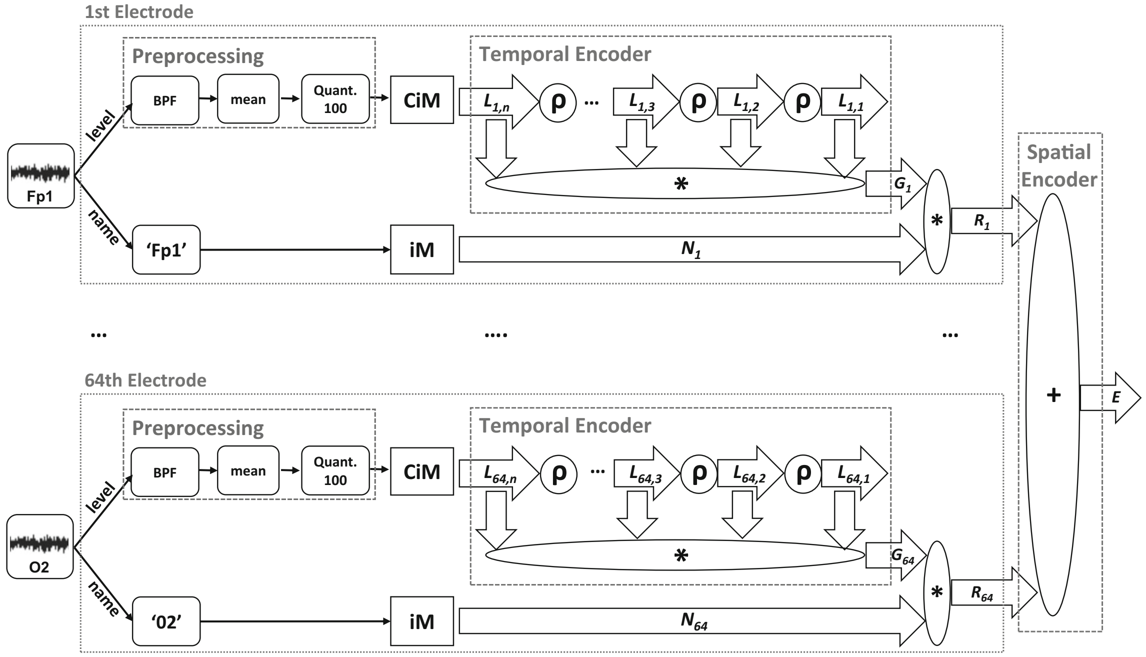


Fig. 2 Temporal–spatial encoder for EEG ERPs: Temporal encoder rotates the signal level to capture its *history* that produces a temporal n-gram (G_i) to be bound with the electrode name (N_i) for constructing a record (R_i); Spatial encoder adds these records across 64 electrodes to produce a temporal–spatial hypervector (E) representing the entire event of ERPs

encode a sequence of n letters to form an n-gram hypervector. By analogy, for example, a sequence of three signal levels of electrode i that are generated at time stamps of 1, 2, and 3 is encoded as follows: the first hypervector L_{i1} is rotated twice $\rho^2 L_{i1}$, the second hypervector is rotated once $\rho^1 L_{i2}$, and finally there is no rotation for the last hypervector L_{i3}. These three hypervectors are then combined with point-wise multiplication into a trigram hypervector, $G_i = \rho^2 L_{i1} * \rho^1 L_{i2} * L_{i3}$. For n-grams at large this becomes:

$$L_{it} = \text{CiM (signal level of } i\text{th electrode at time stamp } t) \quad (2)$$

$$G_i = \prod_{t=1}^{n} \rho^{n-t} L_{it} \quad (3)$$

With this temporal encoding algorithm, one important step is to determine the proper size of an n-gram to be able to capture the entire event of an ERP for the subjects. In this regard, we measured the number of samples available in an event of ERP. Table 1 shows the time windows in which the events of ERP are observed during the trials of 2000 ms for each subject. As shown, this time window can be as large as 600 ms (for S4) containing 308 samples with sampling rate of 512 Hz. Computing an n-gram of this size is inefficient; earlier, we overcame such issue for the EMG signals by highly downsampling the electrode such that the hand gestures can be fit into n-grams where $n \in [3, 5]$ [21]. Similarly, the electrical activity on the EEG electrodes was downsampled to 64 Hz [4]. Consequently, we partition the time widow of ERP to a set of nonoverlapping slices of equal length. The length of every slice is 8 samples as the ratio of sampling rate (512 Hz) to the downsampling rate (64 Hz). With 8 samples in a slice, the number of slices in the window is proportional to the duration of the window, and varies from 16 slices to 29 slices.[2] Then, we take the mean value of samples in the slice, apply the scaling and quantization step and use it as the quantized signal level for CiM. Therefore, the event of ERP can be represented by an n-gram where $n \in [16, 29]$. The last column in Table 1 lists the exact size of n-gram used for each subject. Please note that the n-gram size is the only parameter of our encoder that we *naturally* set it per subject solely based on the duration of ERP event. This is in sharp contrast with other learning methods as we do not rely on inefficient or biologically implausible algorithm for parameter tuning and optimizations.

Our temporal encoder accepts the sequence of n quantized signal levels from the ith electrode and computes the n-gram hypervector G_i. G_i represents the temporal activities of the signal levels of the ith electrode.

[2] For S4, we double the length of slices (16 samples in each) that results in 19 slices to cover the window instead of 38.

4.2 Temporal–spatial encoder: adding temporal hypervectors of 64 electrodes

The 64 temporal hypervectors G_i for the electrodes are encoded into a single temporal–spatial hypervector E for the event in a manner analogous to the holistic record of [14]. The electrode name is treated as a field, and its signal is treated as a value for the field. The channel names $N_1, N_2, \ldots, N_{64}$ are hypervectors that reside in the item memory, iM. They are random, independent, and approximately orthogonal to one another. The signal G_i is bound to its channel name N_i with point-wise multiplication and is represented by the hypervector

$$R_i = G_i * N_i \tag{4}$$

and the entire event is represented by a hypervector E that is the sum of 64 hypervectors for the field–value pairs:

$$E = \sum_{i=1}^{64} R_i \tag{5}$$

The structure of the encoder is shown in Fig. 2.

4.2.1 Class prototypes

The classifier construction is purely based on native operations of HD computing without the involvement of any optimization procedure. The approach is based on the notion of a class prototype. The class prototype is a hypervector representing the entire class. The number of class prototypes equals the number of classes in ERP. Hence, we generate two hypervectors C and W as the class prototypes for ERP: C represents the "correct" class and W represents the "error" (wrong) class. The hypervector C is computed by adding all the hypervectors produced by the temporal–spatial encoder from the "correct" events of ERP. Hence, the addition operation bundles E_j observed in training trial j to a single hypervector C as follows:

$$C \mathrel{+}= E_j \,|\, \cos(C, E_j) < 0.5 \tag{6}$$

Before adding a new event E_j to C, we check whether this event is already in C. This checking forms a conditional addition that adds E_j to C when $\cos(C, E_j) < 0.5$. If C has a high cosine similarity (≥ 0.5) with E_j, it means that the event is already in C, hence there is no need to add the redundant event. Similarly, the hypervector W is computed for the "error" events. Figure 3 shows the procedure of generating the class prototypes. As shown, this procedure is quite simple and can be reused in *online learning*, providing a rule for enabling the class prototype updates.

4.3 Single-trial HD classification

After training, C and W hypervectors are stored in the associative memory as the *learned* patterns of ERP. The same

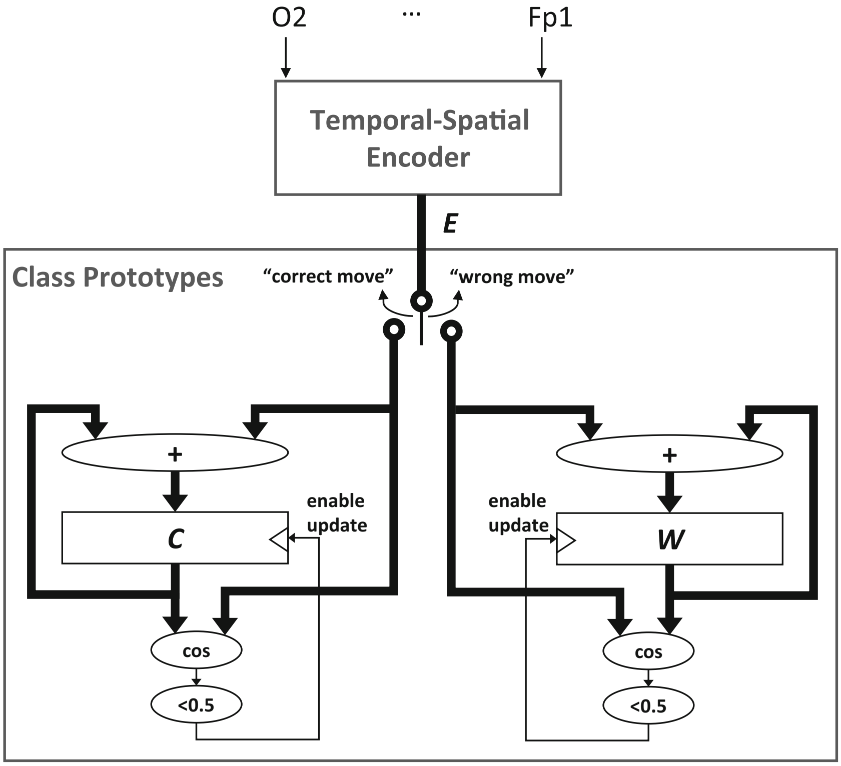


Fig. 3 Constructing class prototypes with the output of temporal-spatial encoder

encoding algorithm is used for both training and testing (i.e., classification). When testing, we call the output of the temporal–spatial encoder a *query* hypervector (Q) since its label is unknown. The query hypervector of a single-trial is then sent to the associative memory to identify its source class. Determining the class of an unknown trial is done by comparing its Q hypervector to all learned hypervectors (i.e., C and W) using the cosine similarity. Finally, the associative memory selects the highest similarity among the two measures and returns its associated label as the class that the Q hypervector has been generated from.

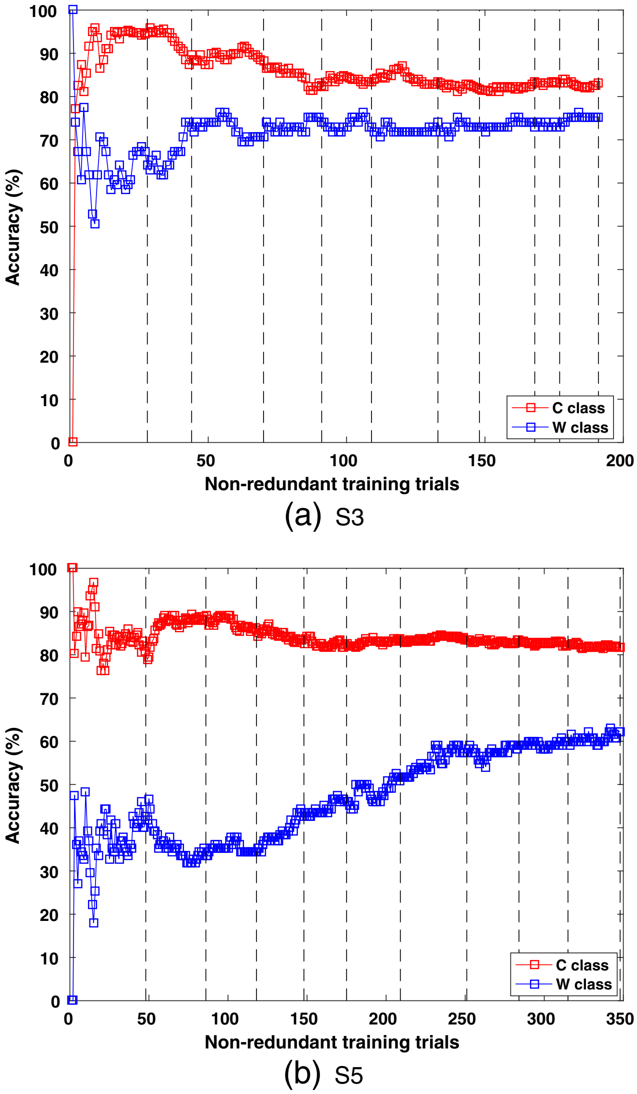
5 Experimental results

In this section, we present experimental results and sensitivity analyzes for classification accuracy of our proposed HD method. We compare it with the baseline method presented in [4] using their dataset available at [2]; session #1 is used for training, and session #2 is used for testing the accuracy. The classification accuracy throughout this paper is measured as macroaveraging that computes a simple average over classes. The macroaveraging gives equal weight to each class, whereas microaveraging gives equal weight to each per-event classification decision. The macroaveraging is a better measure of effectiveness with small classes (the size of "correct" class is $\approx 4\times$ of the "wrong" class). Our HD encoding algorithm and classifier are developed in MATLAB.

5.1 Fast and one-shot learning

Here, we assess how fast the HD training can be done while maintaining high classification accuracy. As described in Section 2.2, the training session is composed of 10 blocks of recording that are shown by vertical dashed lines in Fig. 4. These consecutive blocks provide a total number of ≈ 640 trials for training per subject. We have observed that only some of these training trials can produce a non-redundant temporal–spatial hypervector for addition to the class prototype (i.e., meeting Eq. 6). For instance, as shown in Fig. 4, the training session contains 191 and 348 non-redundant trials for S3 and S5, respectively. Hence, during the training session, every time that we encounter a new non-redundant ERP event, we update the associative memory and measure the classification accuracy for the entire test set.

Figure 4a shows the classification accuracy of both classes for S3. For the very first trials the associative memory is almost empty, but as we encounter new trials it will be lightly populated leading to an increase in the microaveraged accuracy. Training with only the first five non-redundant trials (2.6% of the total), the accuracy of HD classifier reaches to 79.3% which is higher than the baseline accuracy (75.9%) using all training trials. Similarly but with a slower learning rate, the HD classifier achieves the accuracy of 66.3% for S5 by using 183 non-redundant trials (53% of the total) as shown in Fig. 4b.



(a) S3

(b) S5

Fig. 4 Classification accuracy of both classes when we increase the size of non-redundant training trials for two subjects: **a** S3 and **b** S5. Dashed lines separate the 10 blocks of training session

We repeat the aforementioned experiment for all subjects and the results are summarized in Fig. 5. We target the classification accuracy of the baseline that is achieved by using all available trials in the training session, one or two electrode(s) (c.f. Table 1), and with the CAR preprocessing technique. We provide the same conditions for the HD classifier,[3] but with fewer training trials, to assess how fast it can reach to the target accuracy. As shown, the HD

[3] Equation 5 for the spatial encoder is limited to one or two electrode(s) and the CAR filter is applied before the BPF.

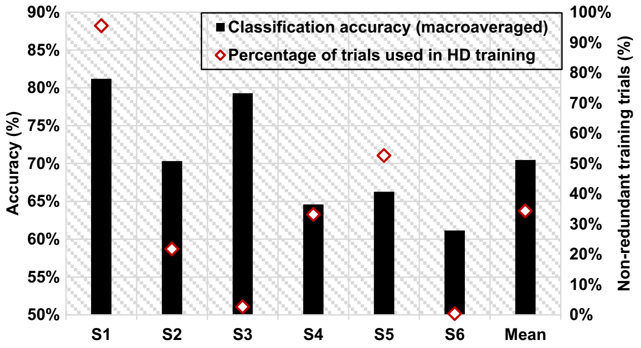


Fig. 5 The percentage of non-redundant training trials that are used to train the HD classier, with resulting accuracy, instead of using all trials

classifier is able to learn faster: it uses only 0.3% of the non-redundant training trials for S6, and up to 96% for S1. On average, the HD classifier reaches the target classification accuracy of 70.5% when trained with only 34% of non-redundant training trials. This translates directly to ≈ 3× faster learning.

5.2 Algorithms for identifying meaningful electrodes

Here, we describe automatic methods to identify useful electrodes. Using the domain expert knowledge, authors in [4] selected FCz, Cz, or both electrodes for their baseline classifier (see Table 1). We observe that these electrodes can be identified as well by two algorithms that are described in the following.

First, we use Fisher scoring algorithm to identify the most important electrodes before mapping them into the HD space. Fisher score, as a filter-based approach, assesses the correlations between features and the class labels to find out features that are efficient for discrimination [9]. It assigns the highest score to the feature on which the data points of different classes are far from each other while requiring data points of the same class to be close to each other. For each electrode, we consider the values of n slices in the time window that form a feature space of $n \times 64$. The electrodes are sorted based on the average Fisher score of their features. Figure 6 (top) shows the top 32 electrodes of the sorted list based on the average Fisher scores of the subjects. As shown, over all the subjects, the electrodes FCz and Cz are among the four most discriminative electrodes.

The second algorithm shows the *analyzability* of the learned HD code. It is inspired by the distribution of distances in HD space. For each electrode, we compute a score that measures the distance between two class prototypes that are generated solely by the electrode. In other words, this score reflects how well a given electrode discriminates between the two class prototypes: the larger, the better. Note that if electrodes i and k receive identical input stimuli, their encoded n-gram hypervectors in Eq. 3 are identical, $G_i = G_k$, because all 64 instances of CiM in Fig. 2 are identical. We compute a hypervector G_i^C for every electrode i by adding the n-gram hypervectors G_i^j over all the trials j belonging to the "correct" class. Similarly, a hypervector G_i^W is computed for the "error" class as shown in Eq. 7.

$$G_i^C = \sum_{j \in \text{Correct}} G_i^j \qquad G_i^W = \sum_{j \in \text{Wrong}} G_i^j \tag{7}$$

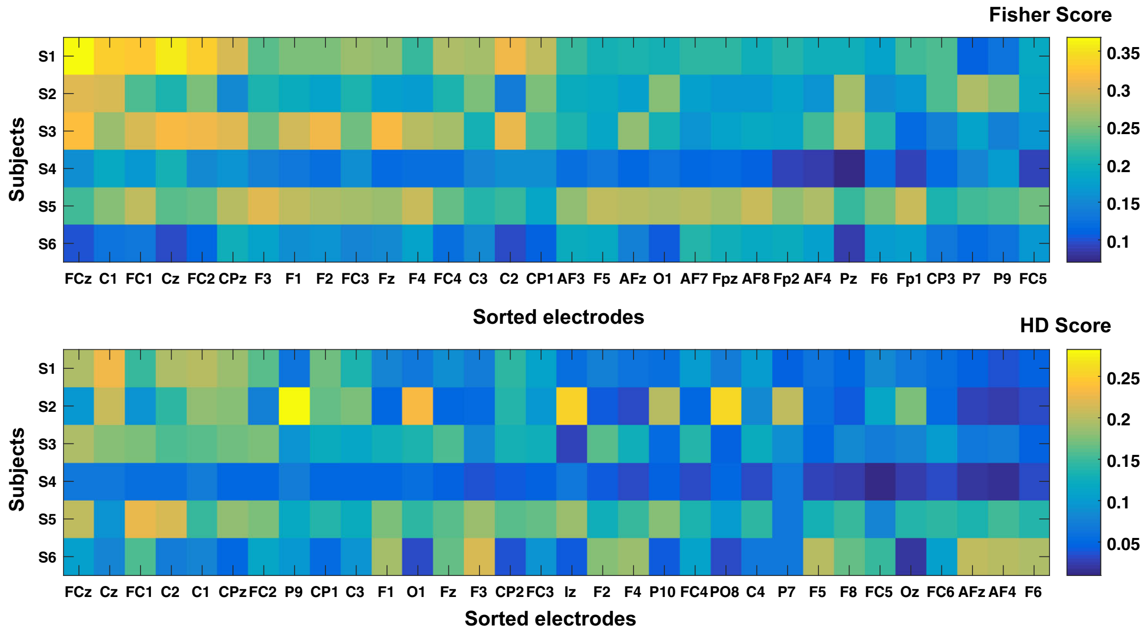


Fig. 6 Sorted electrodes, the top 32, based on: Fisher score (Top); Score in HD space by analyzing the learned code using Eq. 8 (Bottom)

For each electrode i, we can then assign a score by measuring the distance between their G_i^C and G_i^W. Equation 8 shows how this score is computed.

$$\text{score}_i = 1 - \cos(G_i^C, G_i^W) \tag{8}$$

Figure 6 (bottom) shows the scores for each electrode and subject. The electrodes are sorted according to the average score over the subjects; only the top 32 electrodes are shown. As shown, the FCz and Cz electrodes have the highest scores for the six subjects on average. However, all subjects do not exhibit the same sensitivity to these two electrodes. For example, S4 does not show a clear distinction between electrodes.

5.3 Blindly using all electrodes without preprocessing

Although the discriminative electrodes can be identified by a domain expert or algorithms, we want to assess the ability of the HD classifier to operate with noisy inputs and electrodes that do not carry meaningful information. Figure 7 compares the classification accuracy of the baseline method with two instances of our HD classifier. The first one has a setup equivalent of the baseline as aforesaid: uses one or two electrode(s) depending on the subjects, applies the CAR preprocessing filter on every electrode before the BPF step, and uses all training trials. As shown in Fig. 7, this instance of HD classifier surpasses the baseline accuracy across the six subjects. The HD classifier exhibits 67.7–82.7% classification accuracy, with an average of 74.5%, which is 5% higher than the baseline with the same conditions.

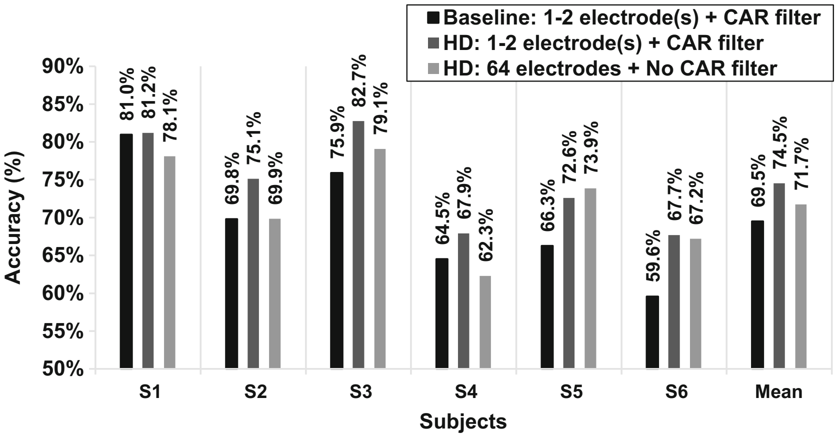
The second instance of our HD classifier operates with the 64 electrodes and without the CAR preprocessing filter. It is illustrated in Fig. 2 where the temporal–spatial encoder accepts the inputs from the 64 electrodes; every electrode signal is immediately passed through a BPF, and its mean is computed over 8 samples followed by the scaling and quantization step before mapping to the HD space by a CiM. As shown, there is no CAR filter in the chain of data. Note that the simple BPF cannot be removed since the ERPs are in the frequency range of 1–10 Hz.

Despite using the 64 electrodes and no CAR filtering, the HD classifier maintains almost the same range of classification accuracy (i.e., 62.3–79.1%) across the six subjects as shown in Fig. 7. This HD classifier shows on average 2.2% higher classification accuracy compared to the baseline. Note that this has been accomplished while this classifier blindly uses largely meaningless electrodes in the encoder regardless of the subjects, while for the baseline authors carefully selected a subset of electrodes per individual subject that can provide meaningful information for the classifier. This also confirms the amenability of HD computing to operate with raw data. In HD computing, the input data is naturally clustered in HD space, and the noise generated by meaningless electrodes tends to cancel out. This desirable property makes it possible to apply HD computing for clustering data with little or no prior knowledge about the nature of the data.

6 Discussion

We have earlier used HD computing to encode biosignals from EMG electrodes for hand gesture recognition [21]. We used a spatial–temporal encoder with small n-gram sizes to deal with EMG recoding from four electrodes. However, encoding EEG data requires larger n-gram sizes which brings up the following issue. In the spatial encoder, the point-wise addition of bipolar hypervectors produces 0s and the point-wise multiplication in the temporal encoder makes them contagious. After a certain size of n, this results in generating an n-gram hypervector where all elements are 0.

Fig. 7 Comparison of microaveraged accuracy of the baseline [4] classifier with two instance of our HD classifier: (1) Using 1 or 2 electrode(s) and CAR preprocessing as in the baseline; (2) Using 64 electrodes without CAR preprocessing



We can start with bipolar hypervectors, meaning that a sum hypervector can have 0s. We can leave the 0s alone, except that when we multiply two hypervectors, we check for 0s and turn them into a $+1$ or a -1 at random. This introduces noise but keeps the hypervectors from eventually turning into all 0s. We will be able to make vectors for n-grams of any length, but the amount of noise will increase with the length of the n-gram. The noise grows as follows. If the noise probabilities of two independent bipolar hypervectors are p_1 and p_2, the noise probability of their product is $p_1 + p_2 - 2p_1p_2$. In contrast, if we let 0s be, they grow faster, at the rate of $p_1+p_2(1-p_1) = p_1+p_2-p_1p_2$. Therefore, in this paper, we chose to change the order of encoders: first doing the temporal encoding of each electrode that produces a bipolar hypervector, and then doing the spatial addition.

7 Conclusion

This paper presents an application of HD computing to the classification of error-related potentials from EEG recordings. Very simple vector-space operations are used to encode analog input signals from 64 electrodes for single-trial classification. The proposed HD classifier requires neither prior knowledge about selecting the electrodes nor extra preprocessing steps. Our HD algorithm blindly encodes the electrical activity of error-related potentials into a temporal–spatial hypervector representing a binary class of the subject's intentions. The learned HD code can be analyzed to identify the most discriminative electrodes. The classification accuracy of our HD classifier for EEG error-related potentials is comparable to a classifier crafted by a skilled professional. Our HD classifier also achieves it with fewer training data.

Acknowledgements This work was supported by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and by Intel Strategic Research Alliance (ISRA) program on Neuromorphic Architectures for Mainstream Computing.

References

1. BNCI Horizon 2020. http://bnci-horizon-2020.eu/
2. Monitoring error-related potentials. http://bnci-horizon-2020.eu/database/data-sets
3. Bertrand O, Perrin F, Pernier J (1985) A theoretical justification of the average reference in topographic evoked potential studies. Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section 62(6):462–464. https://doi.org/10.1016/0168-5597(85)90058-9. http://www.sciencedirect.com/science/article/pii/0168559785900589
4. Chavarriaga R, Millán JDR (2010) Learning from eeg error-related potentials in noninvasive brain-computer interfaces. IEEE Trans Neural Syst Rehabil Eng 18(4):381–388.
5. Ferrez P, Millán JDR (2007) Error-related EEG potentials in brain-computer interfaces. Ph.D. thesis, STI, Lausanne. https://doi.org/10.5075/epfl-thesis-3928
6. Ferrez PW, Millán JDR (2005) You are wrong!—automatic detection of interaction errors from brain waves. In: Proceedings of the 19th international joint conference on artificial intelligence
7. Ferrez PW, Millán JDR (2008) Error-related eeg potentials generated during simulated brain-computer interaction. IEEE Trans Biomed Eng 55(3):923–929. https://doi.org/10.1109/TBME.2007.908083
8. Gayler RW (1998) Multiplicative binding, representation operators & analogy. Advances in analogy research
9. He X, Cai D, Niyogi P (2005) Laplacian score for feature selection. In: Proceedings of the 18th international conference on neural information processing systems, NIPS'05. MIT Press, Cambridge, pp 507–514. http://dl.acm.org/citation.cfm?id=2976248.2976312
10. Hsu J (2014) Ibm's new brain [news]. IEEE Spectr 51(10):17–19. https://doi.org/10.1109/MSPEC.2014.6905473
11. Joshi A, Halseth J, Kanerva P (2017) Language geometry using random indexing. In: de Barros JA, Coecke B, Pothos E (eds) 10th international conference on quantum interaction, QI 2016. Springer, pp 265–274
12. Kanerva P (1988) Sparse distributed memory. MIT Press, Cambridge
13. Kanerva P (2009) Hyperdimensional computing: an introduction to computing in distributed representation with high-dimensional random vectors. Cogn Comput 1(2):139–159. https://doi.org/10.1007/s12559-009-9009-8
14. Kanerva P (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, pp 2–6
15. Kanerva P (2014) Computing with 10,000-bitwords. In: Proceedings of the 52nd annual allerton conference on communication, control, and computing
16. Kanerva P, Kristoferson J, Holst A (2000) Random indexing of text samples for latent semantic analysis. In: Proceedings of the 22nd annual conference of the cognitive science society. Erlbaum, pp 103–6
17. Li H et al (2016) Hyperdimensional computing with 3d vrram in-memory kernels: device-architecture co-design for energy-efficient, error-resilient language recognition. IEEE International Electron Devices Meeting
18. McFarland DJ, McCane LM, David SV, Wolpaw JR (1997) Spatial filter selection for eeg-based communication. Electroencephalogr Clin Neurophysiol 103(3):386–394. https://doi.org/10.1007/s12559-009-9009-8. http://www.sciencedirect.com/science/article/pii/S0013469497000222
19. Najafabadi FR, Rahimi A, Kanerva P, Rabaey JM (2016) Hyperdimensional computing for text classification. Design, automation test in europe conference exhibition (DATE), University Booth. https://www.date-conference.com/system/files/file/date16/ubooth/37923.pdf
20. Nishihara K, TAYA N, KANOH T (2016) A consideration of realizing the brain inspired computer. In: Proceedings of the 9th EAI international conference on bio-inspired information and communications technologies (formerly BIONETICS), BICT'15. ICST, Brussels, pp 495–496. https://doi.org/10.4108/eai.3-12-2015.2262461
21. Rahimi A, Benatti S, Kanerva P, Benini L, Rabaey JM (2016) Hyperdimensional biosignal processing: a case study for emg-based

hand gesture recognition. In: IEEE international conference on rebooting computing

22. Rahimi A, Kanerva P, Millán JDR, Rabaey JM (2017) Hyperdimensional computing for noninvasive brain–computer interfaces: blind and one-shot classification of eeg error-related potentials. In: 10th EAI international conference on bio-inspired information and communications technologies (BICT)
23. Rahimi A, Kanerva P, Rabaey JM (2016) A robust and energy efficient classifier using brain-inspired hyperdimensional computing. In: 2016 IEEE/ACM international symposium on low power electronics and design (ISLPED)
24. Sahlgren M (2005) An introduction to random indexing. In: In methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering, TKE 2005
25. Widdows D, Bing M (2014) Reasoning with vectors: a continuous model for fast robust inference. In: Logic journal of the IGPL