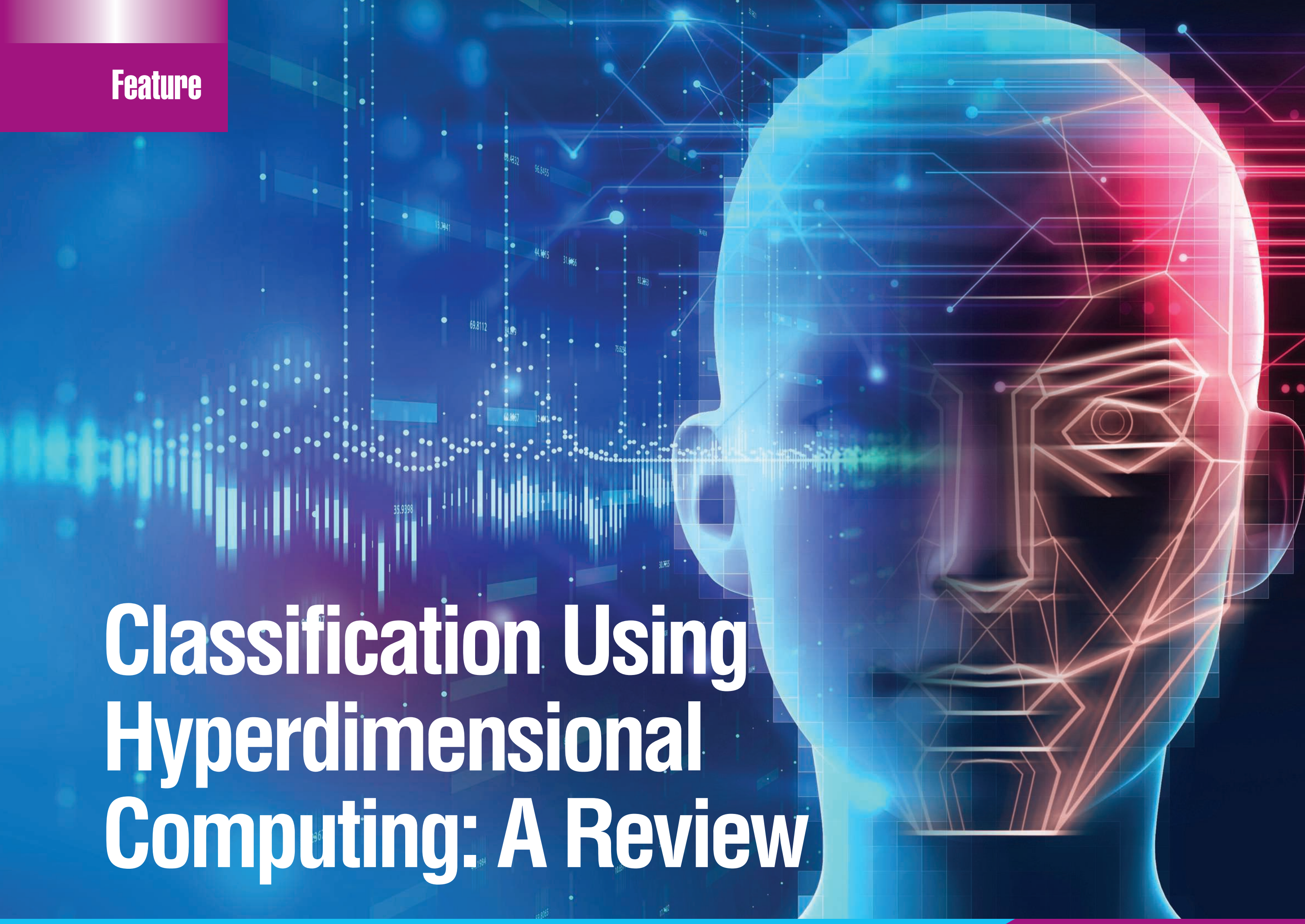

Classification Using Hyperdimensional Computing: A Review

Lulu Ge, *Student Member, IEEE*, and Keshab K. Parhi, *Fellow, IEEE*

Abstract

Hyperdimensional (HD) computing is built upon its unique data type referred to as *hypervectors*. The dimension of these hypervectors is typically in the range of tens of thousands. Proposed to solve cognitive tasks, HD computing aims at calculating similarity among its data. Data transformation is realized by three operations, including addition, multiplication and permutation. Its ultra-wide data representation introduces redundancy against noise. Since information is evenly distributed over every bit of the hypervectors, HD computing is inherently robust. Additionally, due to the nature of those three operations, HD computing leads to fast learning ability, high energy efficiency and acceptable accuracy in learning and classification tasks. This paper introduces the background of HD computing, and reviews the data representation, data transformation, and similarity measurement. The orthogonality in high dimensions presents opportunities for flexible computing. To balance the tradeoff between accuracy and efficiency, strategies include but are not limited to encoding, retraining, binarization and hardware acceleration. Evaluations indicate that HD computing shows great potential in addressing problems using data in the form of letters, signals and images. HD computing especially shows significant promise to replace machine learning algorithms as a light-weight classifier in the field of internet of things (IoTs).

Digital Object Identifier 10.1109/MCAS.2020.2988388
Date of current version: 3 June 2020

I. Introduction

The emergence of hyperdimensional (HD) computing is based on the cognitive model developed by Kanerva [1]. HD computing grew out of cognitive science in answer to the binding problem of connectionist (neural-net) models. When variables and their values are superposed over the same vector, representing which value is associated with which variable requires a formal model. This was initially solved using tensor product variable binding by Smolensky [2] and later by Plate [3] using holographic reduced representation (HRR). The advantage of HRR over tensor product is that it keeps vector dimensionality constant. Systems based on these representations go by many names: HRR, HD, binary spatter code (BSD) [4], binary sparse distributed code (BSDC) [5], multiply-add-permute (MAP) [6], vector symbolic architecture (VSA) [7], and semantic pointer architecture. All rely on high dimensionality, randomness, abundance of nearly orthogonal vectors and computing in superposition.

Instead of computing traditional numerical values, HD computing performs cognition tasks—such as face detection, language classification, speech recognition, image classification, etc.—by representing different types of data using hypervectors, whose dimensionality is in the thousands, e.g., 10,000-*d*, where *d* refers to dimensionality. The human brain contains about 100 billion neurons and 1000 trillion synapses; therefore all possible *states* of a human brain can be described by a high-dimensional vector. In that sense, HD computing is a form of brain-inspired computing. Randomly or pseudo-randomly defined, these hypervectors are composed of independent and identically distributed (*i.i.d.*) components, which can be binary, integer, real or complex [8]. As a brain-inspired computing model, HD computing is robust, scalable, energy efficient and requires less time for training and inference [9]. These features are a result of its ultra-wide data representation and underlying mathematical operations. One thing that should be emphasized is the concept of *orthogonality* of the hypervectors.

The remainder of this paper is organized as follows. Section II presents the background on HD computing, including the data representation, data transformation and similarity measurement. Section III illustrates the general methodology in HD computing and its applicability in learning and inference tasks. Then two common encoding methods to form hypervectors from the input data are presented, and strategies to improve accuracy and/or efficiency are pointed out. Some classical applications as well as several sophisticated designs are reviewed in Section IV. Possible future directions of HD computing are also pointed out in this section. Finally, Section V concludes the paper.

II. Background on HD Computing

In this section, we review HD computing and present a comparison between HD and classical computing. We also describe the similarity metrics for hypervectors and typical mathematical operations used in HD computing.

A. Classical Computing vs HD Computing

Data representation, data transformation and data retrieval play an important role in any computing system. To be more specific, classical computing deals with bits. Each bit is 0 or 1. This can be realized by the absence or presence of electric charge. In terms of computation, data transformation is inevitable. The arithmetic/logic unit (ALU) computes new data using logical operation and four arithmetic operations, including addition, subtraction, multiplication and division [10]. The main memory allows the data to be written and read. Compared to classical computing, HD computing employs hypervectors as its data type, whose dimensionality is typically in the thousands. These ultra-wide words introduce redundancy against noise, and are, therefore, inherently robust.

To transform data, HD computing performs three operations: multiplication, addition and permutation. HD computing transforms the input hypervectors, which are pre-stored in the item memory to form associations or connections. In a classification problem, the hypervectors associated with classes are trained during training process. During the testing process, the test hypervectors are compared with the class hypervectors. The hypervectors generated from training data are referred to as *class* hypervectors and are stored in the associative memory, while those generated from the test data are referred to as *query* hypervectors. An associative search is performed to make a prediction as to which class a given query hypervector most likely belongs. A comparison between the classical and HD computing paradigms is summarized in Table 1. Traditional classification

Table 1.
Comparisons between classical computing and HD computing for classification.

Computing Paradigm	Classical Computing	HD Computing
Data Type	Bit	Hypervector
Data Transformation	Addition, Multiplication, Logic	Add-Multiply-Permute
Storage	Memory	Item Memory, Associative Memory
Training	Weights	Class Hypervectors
Testing	Run Pre-trained Classifier	Associate Query Hypervectors with Class Hypervectors
Model Complexity	High	Low
Accuracy	Very High	Acceptable
Feature Extraction	Easy	Difficult
Number of Features	Many	One

Lulu Ge and Keshab K. Parhi are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA (e-mail: ge000567@umn.edu, parhi@umn.edu)

methods achieve high accuracy using complex models. Training these models typically takes longer time and requires more energy consumption. The models in HD classification are simpler and can be trained in less time with high energy efficiency. However, their accuracy is acceptable, though not as high as traditional models. This is because the accuracy is dependent on feature encoding which is not as well understood as traditional classification.

B. Data Representation

Data points of HD computing correspond to hypervectors—vectors of bits, integers, real or complex numbers. These are roughly divided into two categories: binary and non-binary. For non-binary hypervectors, bipolar and integer hypervectors are more commonly employed. Generally speaking, non-binary HD algorithms achieve higher accuracy, while the binary counterpart is more hardware-friendly and has higher efficiency (see also [11]).

C. Similarity Measurement

As shown in Table 2, two common similarity measurements are adopted in the existing HD algorithms, namely, cosine similarity and Hamming distance. Other similarity measures include dot product (e.g., in MAP) and overlap (e.g., in BSDC).

Table 2.
Similarity measurements in HD computing.

Measurement	Similar	Orthogonal
Hamming distance	0	0.5
Cosine similarity	1	0

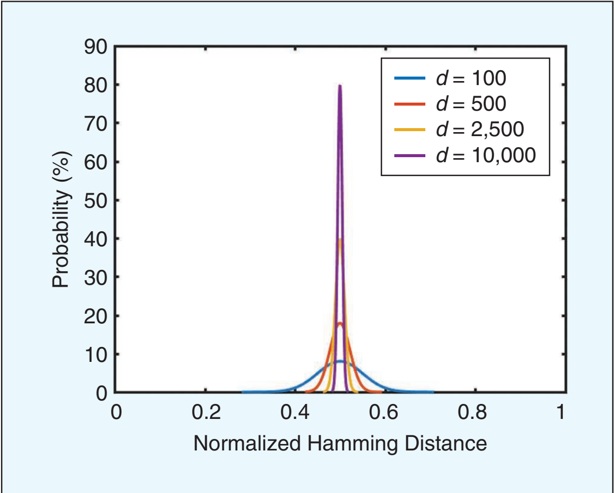

Figure 1. Orthogonality in high dimensions [1, 13, 14].

For non-binary hypervectors, *cosine similarity*, defined by Eq. (1), is used to measure their similarity, focusing on the angle and ignoring the impact of the magnitude of hypervectors, where $|\cdot|$ denotes the magnitude. Unlike the inner product operation [12] of two vectors that affects magnitude and orientation, the *cosine similarity* only depends on the orientation. In most HD algorithms with non-binary hypervectors, cosine similarity is more often used than inner product. Moreover, when $\cos(A, B)$ is close to 1, this implies an extremely high level of similarity. For example, $\cos(A, B) = 1$ indicates two hypervectors A and B are identical. When they are at right angle, then $\cos(A, B) = 0$, and the two orthogonal vectors are considered dissimilar.

$$\cos(A, B) = \frac{A \cdot B}{|A\|B|} \tag{1}$$

For binary hypervectors with dimensionality d, whose components are either 0 or 1, normalized Hamming distance calculated in Eq. (2) is used to measure their similarity [8]. When the Hamming distance of two hypervectors is close to 0, then they are defined as similar. For example, $\text{Ham}(A,B) = 0$ indicates every single bit is same at each position, and A and B are identical. When $\text{Ham}(A,B) = 0.5$, A and B are orthogonal or dissimilar. $\text{Ham}(A,B) = 1$ when A and B are diametrically opposed.

$$\text{Ham}(A,B) = \frac{1}{d}\sum_{i=1}^{d} 1_{A(i) \neq B(i)} \tag{2}$$

One thing that should be emphasized is orthogonality in high dimensions. To put it simply, the randomly generated hypervectors are nearly orthogonal to each other when the dimensionality is in the thousands. Take binary hypervectors as an example. Assume X and Y are chosen independently and uniformly from $\{0,1\}^d$ and the probability p of any bit being 1 is 0.5. Then $\text{Ham}(X,Y)$ is binomially distributed. Fig. 1 shows the probability density function (PDF) of $\text{Ham}(X,Y)$ for 15,000 pairs of randomly selected binary vectors with different dimensions d. As d increases, more vectors become orthogonal. Such orthogonality property is of great interest because orthogonal hypervectors are dissimilar. Moreover, operations performed on these orthogonal hypervectors can form associations or relations.

D. Data Transformation

Three types of operations, add-multiply-permute, are employed in HD computing. The inverse operation for multiplication is also referred to as *release* [14]. The release operation is also used to denote inverse addition. Each operation processes and generates d-dimensional hypervectors. In the following, we illustrate examples of data transformations using binary hypervectors. Without doubt, data transformation can also be employed to non-binary hypervectors, which is in essence similar to the manipulations over binary hypervectors. The only difference is from the point of hardware; for binary hypervectors, the pointwise multiplication can be realized by an exclusive or (XOR) gate.

1) Addition

Pointwise addition, also referred to as *bundling*, computes a hypervector Z using Eq. (3) from the input hypervectors $\{X_1, X_2, \ldots, X_n\}$. Compared to random hypervectors, the generated Z is maximally similar to the n inputs $X_1, X_2, \ldots, X_n$, i.e., Hamming distance between Z and any of the n inputs is at a minimum.

$$Z = [X_1 + X_2 + \ldots + X_n], \tag{3}$$

where $[\cdot]$ indicates the sum hypervector Z is thresholded and binarized to $\{0,1\}^d$ based on the *majority rule*. For convenience, Eq. (4) shows an example for the pointwise addition of three 10-bit binary vectors.

$$\begin{array}{r} A = 0\,0\,0\,0\,1\,1\,0\,0\,1\,1, \\ B = 1\,0\,1\,1\,0\,0\,0\,1\,0\,1, \\ C = 0\,0\,1\,0\,1\,0\,1\,1\,0\,1, \\ \hline [A+B+C] = 0\,0\,1\,0\,1\,0\,0\,1\,0\,1. \end{array} \tag{4}$$

Generally speaking, the addition over odd number of hypervectors has no ambiguity, whereas the addition over an even number can favor either 0 or 1 using the majority function defined in Eq. (5). However, this approach may lead to a biased result for adding two hypervectors. Therefore, the bias in adding even number of hypervectors is usually reduced by adding an extra random vector [15]. Fig. 2 illustrates addition of 10,000-dimensional random hypervectors repeated for 3,000 times. Comparing Fig. 2(b) to Fig. 2(c), we see that specifying in favor of 0 or 1 has little impact over addition. It can be observed from Fig. 2 that the sum is nearly equally similar to the input operands.

$$\text{Majority}(p_1, \ldots, p_n) = \begin{cases} \left\lfloor \dfrac{1}{2} + \dfrac{\left(\sum_{i=1}^{n} p_i\right) - \frac{1}{2}}{n} \right\rfloor, & \text{favor } 0, \\ \left\lfloor \dfrac{1}{2} + \dfrac{\sum_{i=1}^{n} p_i}{n} \right\rfloor, & \text{favor } 1. \end{cases} \tag{5}$$

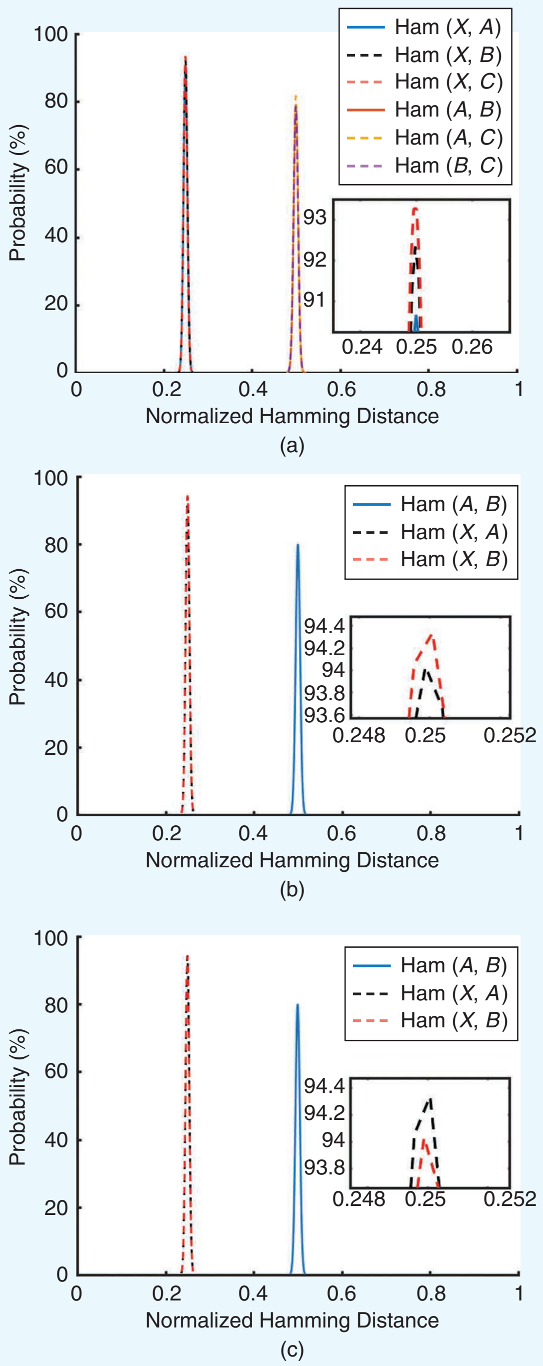

Figure 2. Hamming distance distribution of addition for 10,000-bit hypervectors over 3000 cases. (a) Addition over odd number of hypervectors; (b) and (c) shows the addition over even number favoring 0 and 1, respectively.
(a) $A + B + C = X$.
(b) $A + B = X$ in favor of 0.
(c) $A + B = X$ in favor of 1.

2) Multiplication

Pointwise multiplication, also called *binding*, aims to form associations between two related hypervectors. A and B are bound together to form $X = A \oplus B$, which is approximately orthogonal to both A and B, where $\oplus$ represents the XOR operation. Eq. (6) shows the pointwise multiplication of two 10-bit binary vectors. In a more general case, as shown in Fig. 3, for two randomly generated 10,000-bit binary hypervectors, their pointwise multiplication result is dissimilar to both of them.

$$\begin{array}{r} A = 0\,0\,0\,0\,1\,1\,0\,0\,1\,1, \\ B = 1\,0\,1\,1\,0\,0\,0\,1\,0\,1, \\ \hline A \oplus B = 1\,0\,1\,1\,1\,1\,0\,1\,1\,0. \end{array} \tag{6}$$

3) Permutation

Permutation ρ is a unique unary operation for HD computing, which shuffles the hypervector, let us say A. The resulting permuted hypervector $\rho(A)$ is quasi-orthogonal to the initial A, i.e., the normalized Hamming distance is close to 0.5. Mathematically, permutation can be realized by multiplying a permutation matrix. As a specific permutation, circular shift is widely employed for its friendly hardware implementation. Eq. (7) shows a circular shift of a 10-bit binary vector with Ham $(A, \rho(A)) = 0.4$. Expected Hamming distance is supposed to be 0.5 for ultra-wide hypervectors. Fig. 4 indicates the permutation result shows dissimilarity with the original 10,000-bit hypervector.

$$\begin{array}{r} A = 0\,0\,0\,0\,1\,1\,0\,0\,1\,1, \\ \hline \rho(A) = 1\,0\,0\,0\,0\,1\,1\,0\,0\,1. \end{array} \tag{7}$$

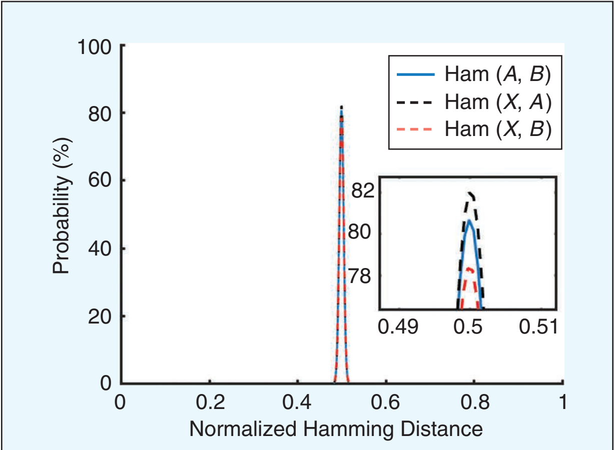

Figure 3. Hamming distance distribution of multiplication $X = A \oplus B$ for 10,000-bit hypervectors over 3000 cases.

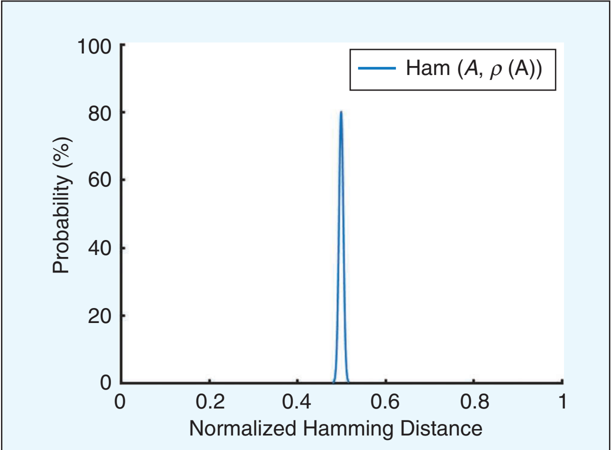

Figure 4. Hamming distance distribution of permutation for 10,000-bit hypervectors over 3000 cases.

Examples. We illustrate applications of above operations. For more details, please refer to [16]. Assume that A, B, C, P, S, X, Y, Z represent 10,000-d random hypervectors:

- *Encode* a pair: To encode "$x = a$", where x is a variable with numerical value same as a, use multiplication to bind their corresponding hypervectors X and A. The encoding is represented by the generated hypervector $P = X \oplus A$.
- *Release* the value from the pair:

$$X \oplus P = \underbrace{X \oplus (X \oplus A)}_{X \oplus X \text{ cancels out}} = A \tag{8}$$

- *Represent* a set: Given the set $s = \{a, b, c\}$, we have

$$S = [A + B + C] \tag{9}$$

- *Encode* a data record: Given a record with a set of bound pairs $d =$ '$(x = a)$ & $(y = b)$ & $(z = c)$,' the record is encoded as:

$$D = [X \oplus A + Y \oplus B + Z \oplus C] \tag{10}$$

- *Extract* the value from a record: To retrieve the value of x:

$$\begin{aligned} A' &= X \oplus D \\ &= \underbrace{X \oplus [X \oplus A + Y \oplus B + Z \oplus C]}_{\text{distributed}} \\ &= X \oplus X \oplus A + X \oplus Y \oplus B + X \oplus Z \oplus C \\ &= \underbrace{X \oplus X \oplus A}_{= A} + \underbrace{(X \oplus Y \oplus B + X \oplus Z \oplus C)}_{\text{noise}} \\ &\approx A \end{aligned} \tag{11}$$

- *Encode* a sequence: Given (a, b), then

$$AB = \rho(A) \oplus B \tag{12}$$

- *Extend* the sequence: Extend (a, b) to (a, b, c) using:

$$\begin{aligned} ABC &= \rho(AB) \oplus C \\ &= \rho(\rho(A)) \oplus \rho(B) \oplus C \end{aligned} \tag{13}$$

- *Extract* the first element of the sequence:

$$\begin{aligned} &\rho^{-1}\rho^{-1}(ABC \oplus BC) \\ &= \rho^{-1}\rho^{-1}(\rho(\rho(A)) \oplus \rho(B) \oplus C \oplus \rho(B) \oplus C) \\ &= A \end{aligned} \tag{14}$$

where ρ^{-1} is the inverse operation of permutation ρ.

III. Learning and Classification by HD Computing

The first wave of using HD for classification started in 1990s [17]–[20]. The current applications of HD for classification can be interpreted as the second wave.

A. The HD Classification Methodology

A system diagram for the classification tasks using HD computing is shown in Fig. 5. In general, 1). during the learning phase, the encoder employs randomly generated hypervectors (pre-stored in the item memory) to map the training data into HD space. A total of k class hypervectors are trained and stored in the associative memory. 2). During the inference phase, the encoder generates the query hypervector for each test data. Then the similarity check is conducted in the associative memory between the query hypervector and every pre-trained class hypervector. Finally, the label with the closest distance is returned.

B Encoding Methods for HD Computing

HD computing can address various types of input data, including letters, signals and images. However, we need to map those input data into hypervectors, and this process corresponds to encoding. The encoding process is somewhat similar to extraction of features. Among the existing HD algorithms, the two encoding methods commonly used include *record-based encoding* and *N-gram-based encoding*. A toy example related speech signals is used for illustration.

Using Mel-frequency cepstral coefficients (MFCCs) [22], the voice information stored in continuous signals can be mapped into the frequency domain. A feature vector with N elements can be obtained. Each element has its feature value, which is evenly discretized or quantized from $\{F_{\min}, F_{\max}\}$ to m different levels.

1) Record-based Encoding

This encoding method employs two types of hypervectors, representing the feature position and feature value, respectively. It may be noted that a variation of record-based encoding based on permutations and a chain of binding operations was proposed in [24]. In this encoding, **position hypervectors** ID_i are randomly generated to encode the feature position information in a feature vector, where $1 \le i \le N$.. The feature value information is quantized to m **level hypervectors** $\{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_m\}$. For an N-dimensional feature, a total of N level hypervectors $\bar{\mathbf{L}}_i$ should be generated, which are chosen from m level hypervectors $\{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_m\}$ based on the feature value. Note that, position hypervectors ID_i are orthogonal to each other, while level hypervectors $\{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_m\}$ are supposed to have correlations between the neighbors. To realize this, in [25] the first level hypervector $\mathbf{L}_1$ represents the feature value $F_{\min}$. Then each time, d/m randomly selected bits are flipped to generate the next level hypervector, where d is the dimensionality of the hypervectors. The continuous bit-flipping was first introduced in [23] and later followed by other use cases [26]–[28]. This bit-flipping approach ensures the correlations between neighbor levels, while the last level hypervector $\mathbf{L}_m$ is nearly orthogonal to $\mathbf{L}_1$. The encoding occurs by binding each position hypervector with its level hypervector. As described in Eq. (15), the final encoding hypervector $\mathbf{H}$ can be obtained by adding these results together. The entire encoding process is illustrated in Fig. 6.

$$\mathbf{H} = \bar{\mathbf{L}}_1 \oplus \mathrm{ID}_1 + \bar{\mathbf{L}}_2 \oplus \mathrm{ID}_2 + \ldots + \bar{\mathbf{L}}_N \oplus \mathrm{ID}_N,$$
$$\bar{\mathbf{L}}_i \in \{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_m\}, \text{ where } 1 \le i \le N. \quad (15)$$

2) N-gram-based Encoding

The method of mapping N-gram statistics into hypervectors was proposed in [30]. First random level hypervectors are generated. Then the feature values are

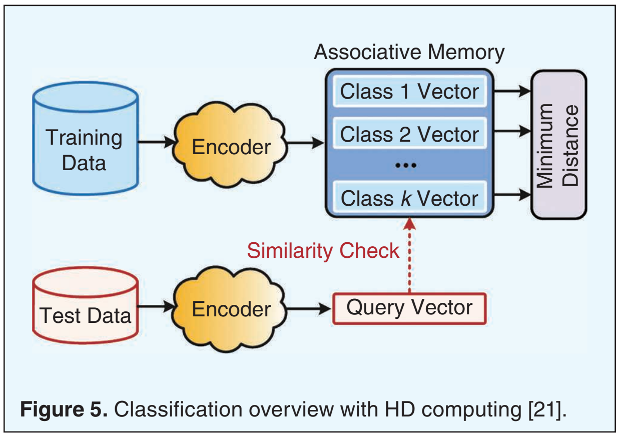

Figure 5. Classification overview with HD computing [21].

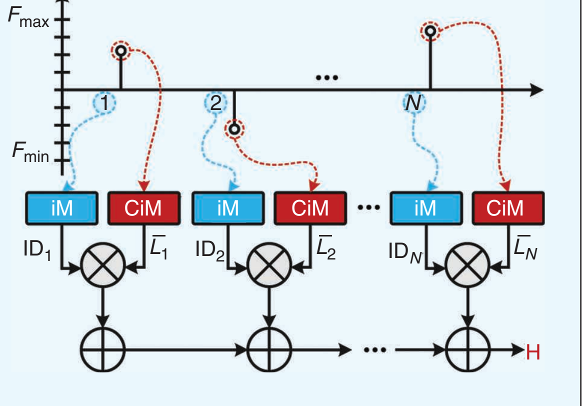

Figure 6. Record-based encoding [23]. Note iM refers to item memory, which stores the position hypervectors, and CiM refers to continuous item memory [13], which stores level hypervectors.

obtained by permuting these level hypervectors in this encoding method. For example, the level hypervector $\bar{\mathbf{L}}_i$ corresponding to the i-th feature position is rotationally permuted by $(i-1)$ positions, where $1 \le i \le N$. We can get the final encoded hypervector $\mathbf{H}$ by Eq. (16). Such an encoding process is illustrated in Fig. 7.

$$\mathbf{H} = \bar{\mathbf{L}}_1 \oplus \rho\bar{\mathbf{L}}_2 \oplus \cdots \oplus \rho^{N-1}\bar{\mathbf{L}}_N, \\ \bar{\mathbf{L}}_i \in \{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_m\}, \text{ where } 1 \le i \le N. \tag{16}$$

Remark *As stated in* [29], *for speech recognition, the N-gram-based encoding method achieves lower accuracy than record-based counterpart. This encoding method is also used to address data types of letters, such as language recognition* [21] *and DNA sequencing* [31].

C. Benchmarking Metrics in HD Computing

In HD computing, there is always a tradeoff between accuracy and efficiency, e.g., see [32]. As shown in Fig. 8, a large amount of work has been carried out to improve the classification accuracy, energy efficiency, or both at the same time.

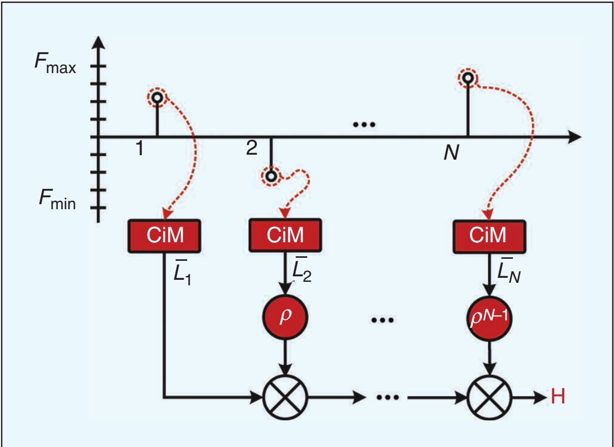

Figure 7. *N*-gram-based encoding [29]. CiM stores level hypervectors which are mutually orthogonal.

1) Accuracy

In terms of accuracy, the encoding method plays a significant role since each encoding may not be efficient for different types of data. Good encoding for HD to achieve high accuracy is hard [33]. In this sense, an appropriate choice of encoding method can improve the accuracy. Efficient encoding approaches have been presented in [34]. The approach in [29] integrates different encoding methods together to achieve higher accuracy at the expense of hardware area. Compared to single-pass training, retraining iteratively improves the training accuracy [28]. Thus the classification accuracy is improved by using a more accurately trained model. Moreover, using binary hypervectors may degrade the accuracy. Hence with enough resources, non-binary models can be used to achieve high accuracy.

2) Efficiency

For efficiency, improvements mainly focus on algorithm and hardware characteristics. From the algorithm perspective, dimension reduction is the most natural way to realize efficiency. Simulations show that slightly reducing the dimensionality of hypervectors, the classification accuracy still remains in an acceptable range but saves hardware resources [25]. Binarization, which refers to employing binary hypervectors instead of non-binary model, accelerates computation and reduces hardware resources [35]. The precision is degraded by quantizing the non-binary HD model. QuantHD has been proposed in [25] to achieve higher efficiency with minimal impact on accuracy. Sparsity was introduced in HD computing in the framework of BSDC [36]. Tradeoff between dense and sparse binary vectors has been presented in [32]. By introducing the concept of sparsity to hypervector representation, [37] proposes a novel platform, SparseHD, which reduces inference computations and leads to high efficiency. From the hardware perspective, HD computing involves a large number of bit-wise operations, as well as the same computation flow for

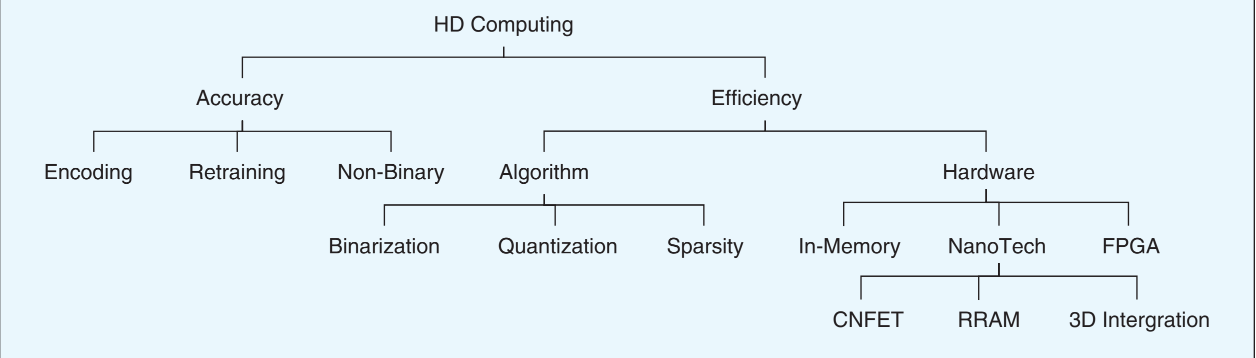

Figure 8. Two benchmarking metrics in HD computing and some possible ways to improve these metrics.

Classical HD computing applications in classification tasks can be categorized based on their input data types, namely letters, signals and images.

different HD applications, making FPGA a nice platform for hardware acceleration [38]. Moreover, as proposed in [39], combining HD computing with the concept of in-memory computing, which is featured as RAM storage and parallel distribution, may create opportunities for HD acceleration. Additionally, several emerging nanotechnologies, including carbon nanotube field-effect transistors (CNFETs) [40], resistive RAM (RRAM) [9], and monolithic 3 D integration [41], have demonstrated implementations of HD computing at high speed [40]. Dimensionality reduction has been evaluated in an actual prototyped system using vertical RRAM (VRRAM) in-memory kernels in [42].

IV. Applications in HD Classification

In what follows, some classical HD computing applications in classification tasks as well as several novel design approaches that can balance tradeoff of accuracy and efficiency are described. They are categorized based on their input data types, namely letters, signals and images.

A. Letters

1) European Language Recognition Using HD Computing

HD computing for European language recognition was first explored by [30]. Literature [40] presents an HD computing nanosystem, which implements HD operations based on emerging nanotechnologies—CNFETs, RRAM and 3 D integration—offering large arrays of memory and resulting in reduction of energy consumption. From its three-letter sequence called *trigrams*, such a nanosystem can identify the language of a given sentence [40]. Define a profile by a histogram of trigram frequencies in the unclassified text. The basic idea is to compare the trigram profile of a test sentence with the trigram profiles of 21 languages, and then find the target language which has the most similar trigram profile [30].

- **Baseline.** Scan through the text and count the trigram to compute a profile. A total of $27^3 = 19,683$ trigrams are possible for the 26 letters and the space. Thus the trigram counts can be encoded into a 19,683-dimensional vector and such vectors can be compared to find the language with the most similar profile. However, this straightforward and simple approach generalizes poorly. Specifically, compared to trigrams, higher-order N-grams will have higher complexity. For example, the number of possible pentagrams is $27^5 = 14,348,907$.
- **HD classification algorithm.** 1). Choose a set of 27 letter hypervectors randomly, serving as the seed hypervector. Note that all training and test data employ the same seeds. In this design, the dimensionality is selected to be 10,000. 2). Generate trigram hypervectors with permutation and multiplication. For example, let (a, b, c) represent a trigram. Then rotate the hypervector A twice, hypervector B once, and use hypervector C with no change, and then multiply them component by component as described in Eq. (13). 3). The target profile hypervector is then the sum of all the trigram hypervectors in the text. 4). Compare the profile of a test sentence to the language profiles, and return the most similar one as the classification result.

Compared to the baseline algorithm, the HD algorithm generalizes better to any N-gram size when 10,000-dimensional hypervectors are used.

The HD classification hardware architecture for language recognition using trigrams proposed in [21] is shown in Fig. 9. Two main modules are implemented. They include the encoding module and the search module. 1). The encoding module takes a stream of letters as the input. Each letter is mapped to the HD space and its corresponding randomly generated hypervector is stored in the item memory. Here it addresses the trigrams where each group of three hypervectors produces a trigram hypervector. Accumulate those trigram hypervectors and perform the majority operation using the threshold to generate a text hypervector. 2). During the training phase, a total of 21 text hypervectors are trained as the learned class hypervectors and are stored in the associative memory in the search module. During the testing phase, the encoding module generates the text hypervector as a query hypervector. This query hypervector is then broadcast to the search module and compared to the stored class hypervectors to predict the language label, which has the closest similarity. As listed in Table 4, the HD classifier achieves 96.70% accuracy.

Using the same architecture shown in Fig. 9, and combining with the emerging nanotechnologies—CNFETs, RRAM and their monolithic 3 D integration—the HD computing hardware implementation achieves classification accuracy up to 98% for over $> 20,000$ sentences [40].

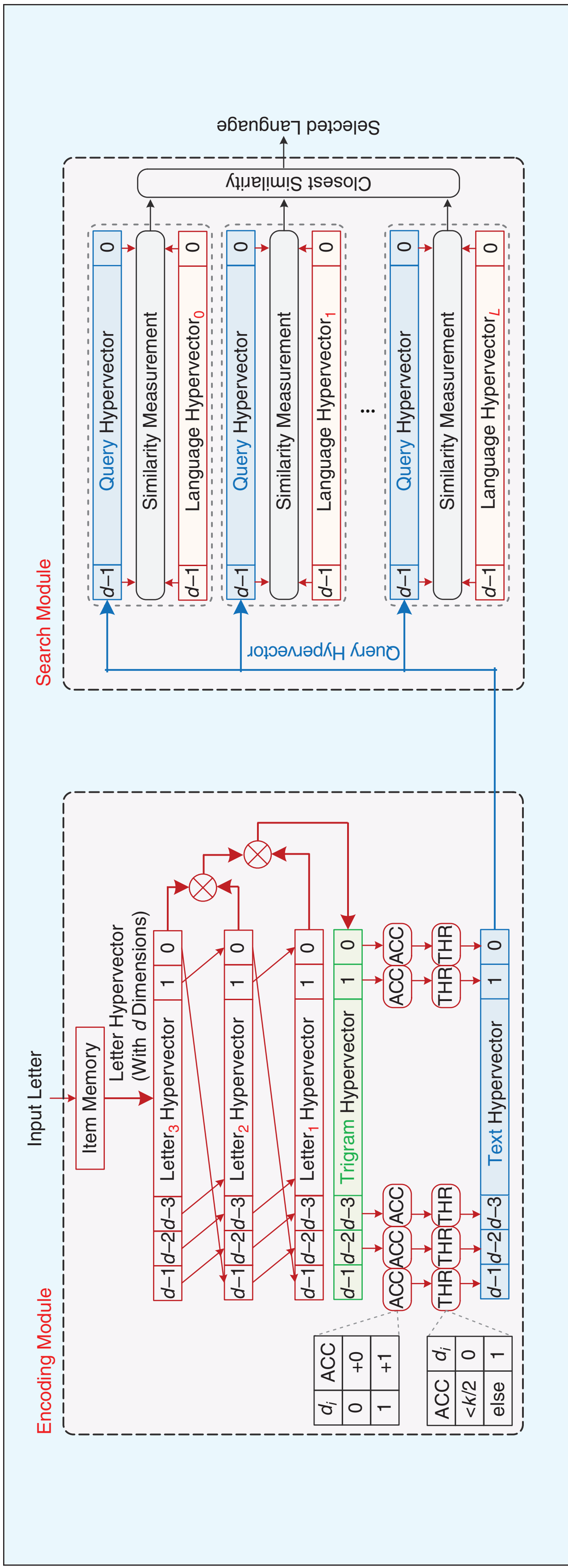

Figure 9. The architecture for language recognition with HD computing [21, 40].

B. Signals

1) HD Classification for Speech Recognition

The development of the Internet of Things (IoT) has motivated the market need for speech recognition. Though deep neural networks (DNNs) have been widely used for speech recognition, it requires expensive hardware and high energy consumption. This has inspired research for speech recognition based on HD computing which can achieve fast computation and energy efficiency.

In [28], VoiceHD, a new speech recognition technique, is proposed for classifying 26 letters from the spoken dataset. At the beginning, the voice signal is transformed to the frequency domain, which contains N frequency ID channels and M levels. Then VoiceHD maps these ID and level information into random hypervectors stored in the item memory. Combining these hypervectors, in the training phase, VoiceHD encoding module generates the learned patterns corresponding to 26 hypervectors that are stored in the associative memory. In the testing phase, VoiceHD uses the same encoding module to generate the query hypervector, which is broadcast to the associative memory. Comparing the query hypervector with the stored 26 class hypervectors, the hypervector with maximum similarity is retrieved to predict the letter. Here, dimensionality d of the hypervectors is 10, 000.

Researchers tested their VoiceHD design over Isolet dataset [44], where a total of 150 subjects spoke the name of each letter of the alphabet twice. The key findings are as follows: 1). Varying the value of M, the number of levels of the amplitude between -1 and 1, with N, the number of frequency bins, fixed at 617, the recognition accuracy increases with increase in M. Note the encoding efficiency degrades with large $M > 10$. The maximum accuracy reaches 88.4% using $M = 10$. 2). To improve the classification accuracy, researchers retrain the associative memory by modifying the trained class hypervectors. The accuracy can be improved to 93.8%. 3). Combining VoiceHD with a small neural network, the corresponding VoiceHD + NN flow is shown in Fig. 10. Such a small NN has three layers. There are 26 neurons in the first layer, 50 neurons in the hidden layer and another 26 neurons in the last layer. The classification accuracy can be improved to be 95.3%. 4). Compared to the pure NN with 93.6% classification accuracy, VoiceHD and VoiceHD+NN show 4.6× and 2.9× faster training speed, 5.3× and 4.0× faster testing speed, and 11.9 × and 8.6× higher energy efficiency, respectively.

In dealing with signals, HD computing usually makes use of floating point models to improve the classification accuracy at the cost of high computation cost.

2) Seizure Detection Using HD Computing

The Laelap algorithm, which utilizes local binary pattern (LBP) codes to conduct the feature extraction from iEEG signals, has been proposed in [43] for seizure prediction. Here HD computing is applied to capture the statistics of the time-varying LBP codes for all the electrodes. Fig. 11 illustrates the complete processing chain. 1). Since the down-sampling frequency is 512 Hz, thus every one second (1 s) data contains 512 samples. Among these samples, the sampled iEEG signals are encoded to 6-bit LBP codes. This completes the feature extraction part. 2). It utilizes record-based encoding, where two types of hypervectors are randomly generated. Specifically, each LBP code is transformed to a d-dimensional hypervector C_i, while the hypervectors E_i are used to represent the corresponding electrode name. For every new sample, the hypervectors E_i and C_i are bound together to form a composite hypervector $S = [C_1 \oplus E_1 + \cdots C_n \oplus E_n]$, where n is the number of electrodes for a specific patient. Then the histogram of LBP codes H is computed for a moving window of 1 s with 0.5 s overlap. Therefore the composite hypervector $H = [S^1 + S^2 + \cdots + S^{512}]$ is updated every 0.5 s. 3). For learning, two prototype hypervectors P_1 and P_2 should be trained. For interictal prototype vector P_1, all H computed over 30 s should be accumulated and normalized to be stored in the associative memory. Depending on the seizure's duration, the ictal prototype vector P_2 is generated using all H over an ictal state, which may last 10 s to 30 s. 4). For classification, comparing P_k with a query H, the label is updated every 0.5 s with the shortest Hamming distance Ham(H, P_k), where $k = 1, 2.$ 5). The algorithm also generates the seizure alarm. In postprocessing, if the last 10 labels all indicate P_2 $(t_c = 10)$ and the distance score $\Delta > t_r$, then the seizure alarm is generated.

The evaluation shows the Laelaps algorithm outperforms other machine learning methods, such as SVM, in terms of energy efficiency. It is worth noting that many simpler seizure detection and prediction algorithms have been proposed in the literature [45]–[49]. A fair comparison of classifier accuracy between HD and traditional classification needs to be explored in future.

3) Quantization in HD Computing

In dealing with signals, HD computing usually makes use of floating point models to improve the classification accuracy at the cost of high computation cost. In [25], QuantHD is proposed as a quantization of HD model, which projects the trained non-binary hypervectors to a binary or ternary model, with elements in $\{0, 1\}$ or $\{-1, 0, +1\}$, to represent class hypervectors. To compensate the accuracy degradation caused by quantization, a retraining approach is used where an iteration number of 30 is pre-defined. The similarity check is no longer cosine metric (non-binary model), but Hamming distance (binary model) or dot product (ternary model). Compared to the existing binarized HD computing, such QuantHD improves on average 17.2% accuracy with a similar computation cost.

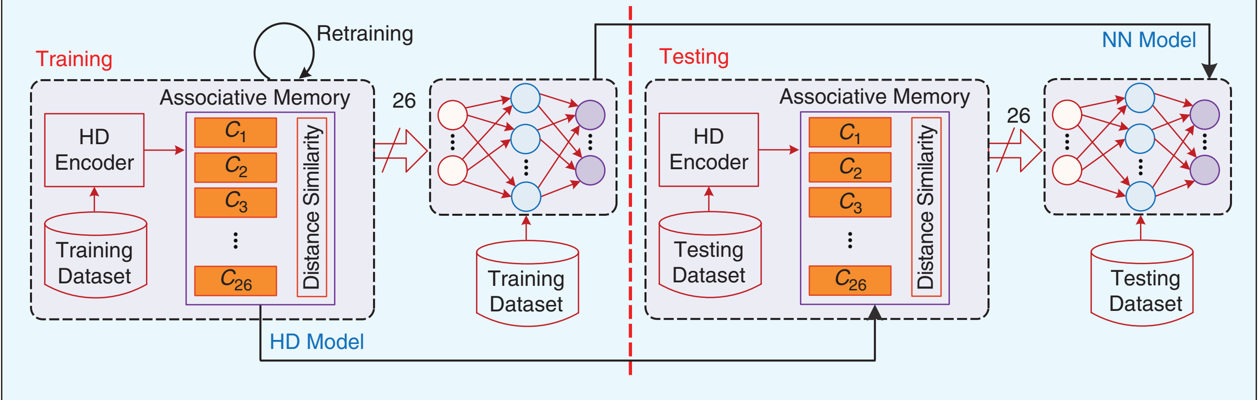

Figure 10. VoiceHD+NN flow for training and testing [28].

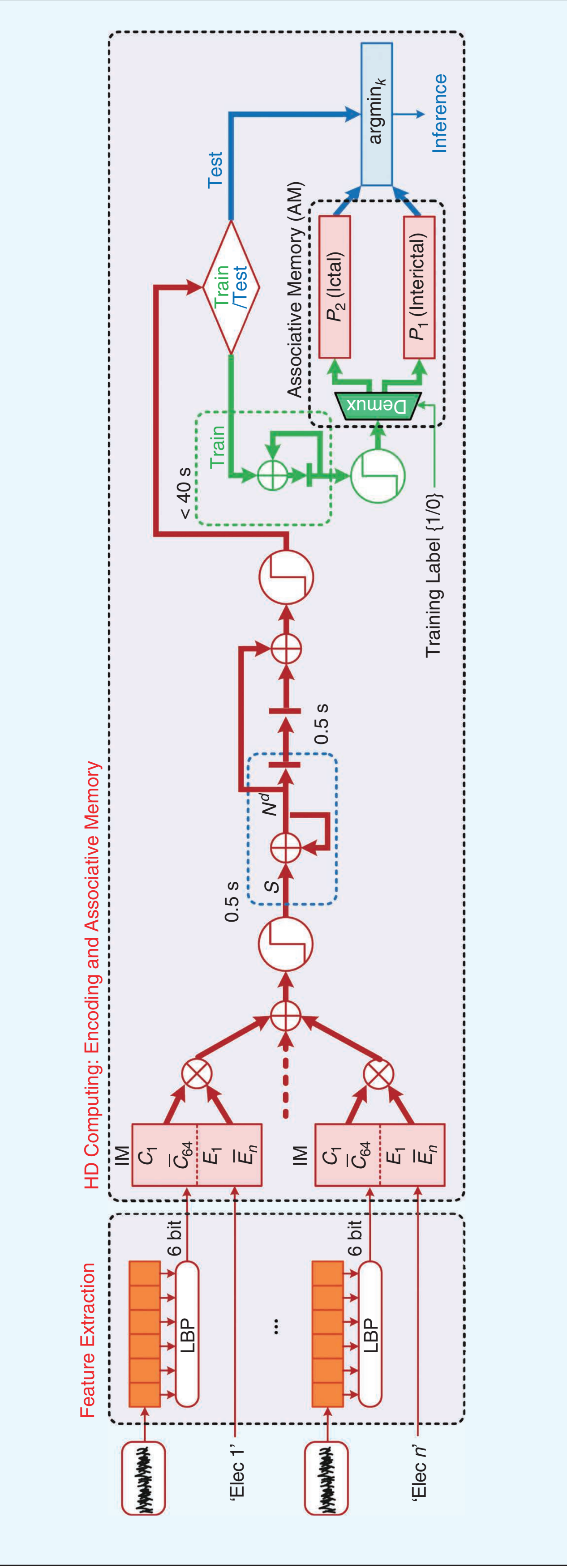

Figure 11. The architecture for Laelaps with HD computing to detect and alarm seizure [43].

4) HD Computing Using Model Compression

As a mathematical framework, HD computing can be an alternative for machine learning problems. This was envisioned in [50]. Due to the high dimensionality, the inference of HD computing is quite expensive, especially when it is applied to the embedded devices with limited resources. For example, the memory is limited. Therefore, reducing the high dimensionality of hypervectors without sacrificing the accuracy has been investigated in [51]. Thus, CompHD is a general method that compresses the model size with the minimal loss of accuracy. The addressed hypervectors are in $\{-1,1\}^d$. Instead of Hamming distance, the similarity metric in CompHD is cosine similarity.

To reduce the HD model size, it is natural to use low dimensional hypervectors. However, experimental results of three practical applications using different dimensionalities in HD classification show that the efficiency is improved by reducing model size at the cost of accuracy.

To maintain high accuracy when reducing the dimensionality, the proposed CompHD employs the architecture shown in Fig. 12. With no reduction in model size, C_i represents the class hypervector, Q represents the query hypervector, where $1 \le i \le k$. In CompHD, class hypervectors and query hypervectors are compressed, which means the original hypervectors are divided into s segments. To store most of the information in original hypervectors with the full size, using Hadamard method [52], CompHD generates P_1, $P_2, \ldots, P_s$, which are in $\{-1,1\}^D$ and are orthogonal to each other, where $D = d/s$. Specifically, the compressed class hypervector C' and query hypervector Q' are calculated using multiplication and addition in HD as described by Eq. (17). By doing so, only little information is lost when we compress the model size, and high accuracy can be maintained.

$$C' = \sum_{i=1}^{s} P_i C^i, \qquad Q' = \sum_{i=1}^{s} P_i Q^i \tag{17}$$

Their evaluation shows that, compared to the original HD classification that purely reduces the dimensionality with the compression factor $s = 20$, the classification accuracy for the three applications is still in an acceptable range. In particular, maintaining the same accuracy as the original, CompHD can on average reduce model size by 69.7% while still achieving 74% energy improvement and 4.1× execution time speedup in the context of activity recognition, gesture recognition and valve monitoring applications [51]. Therefore,

Single-pass training leads to low accuracy. To improve this, iterative training might be one efficient solution. However, a lack of controllability of training iterations in HD classification may result in slow training or divergence.

CompHD is suitable for low-power IoT devices to achieve higher efficiency with a comparable accuracy.

5) Adaptive Efficient Training for HD Computing

Single-pass training leads to low accuracy. To improve this, iterative training might be one efficient solution. However, a lack of controllability of training iterations in HD classification may result in slow training or divergence. To solve this training issue, [54] proposes a retraining approach, AdaptHD.

The basic idea is illustrated as follows: 1). Conduct the initial training by using binary hypervectors to generate the non-binary class hypervectors. 2). Retrain the class hypervectors by looking at the similarity of each trained class hypervectors (C) with the training hypervector (H). Update the model using Eq. (18) if the current training hypervector leads to a misclassification error. Otherwise there is no change. For example, there is a mismatch if H_i is supposed to belong to C_{correct} but is classified as C_{wrong}, where C_{correct} and C_{wrong} denote different class hypervectors and H_i represents the ith training hypervector. 3). After convergence, which means the last three iterations of retraining show less than 0.1% accuracy change, then binarize the final trained model for inference.

$$\begin{cases} C_{\text{wrong}} = C_{\text{wrong}} - \alpha H_i, \\ C_{\text{correct}} = C_{\text{correct}} + \alpha H_i. \end{cases} \tag{18}$$

Insights are gained by their results: 1). Small α needs more iterations to get the near best accuracy. The smooth curve indicates small α is better for fine-tuning. 2). Large α gets to the near best accuracy much faster, but its high fluctuation may lead to divergence. Based on these two findings, AdaptHD uses large α first to get the near best accuracy faster, then changes to smaller α for fine-tuning until convergence. This is similar to adjusting the step size in the normalized least mean square (LMS) algorithm [55]. AdaptHD offers three types of adaptive methods:

- Iteration-dependent AdaptHD. The change of value α depends on iterations. In the beginning, α starts with a large $\alpha_{\max}$. The learning rate α changes based on the average error rate in the previous β iterations. If error rate decreases, indicating convergence, then use smaller α; otherwise, increase α.
- Data-dependent AdaptHD. The value α differs in a certain iteration for all data points, and it changes depending on the similarity of the data point with the class hypervectors. Large distance uses large α to reduce the difference.
- Hybrid AdaptHD. Combining the two models, hybrid AdaptHD can achieve high accuracy as iteration-dependent AdaptHD and fast speedup as data-dependent AdaptHD.

The evaluation shows that, compared to the existing HD algorithm, their hybrid AdaptHD can achieve 6.9× speedup and 6.3× energy-efficiency improvement.

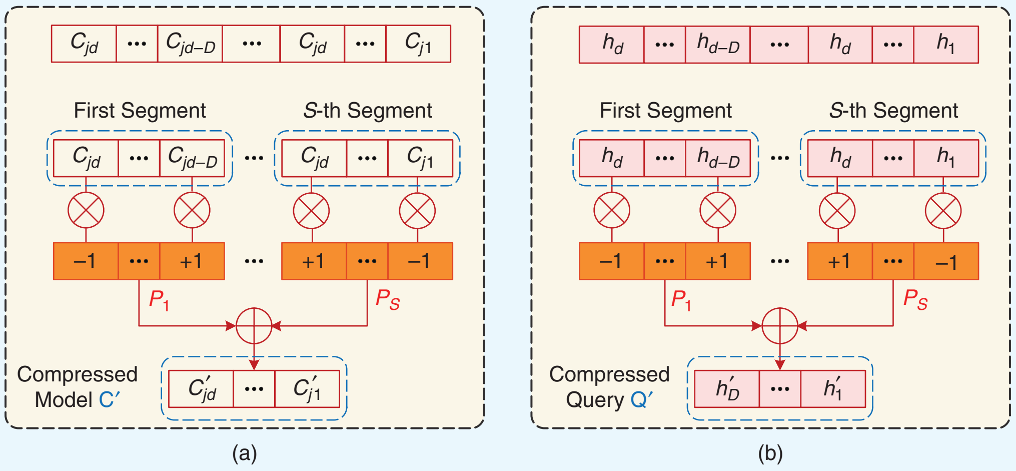

Figure 12. CompHD for (a) an HD model and (b) a query data [51].

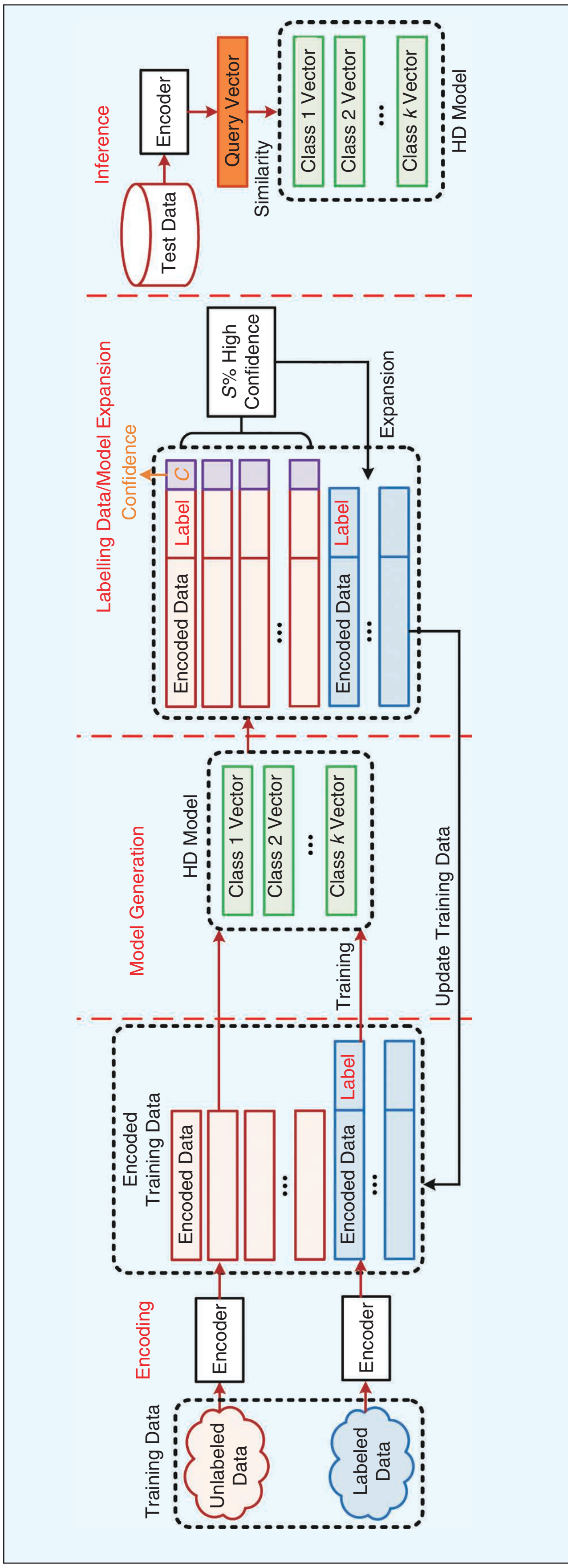

Figure 13. Overview of SemiHD framework supporting self-training in HD space [53].

6) A Binary Framework for HD Computing

Generally speaking, HD classification using binary hypervectors shows lower accuracy but higher energy efficiency than non-binary ones. This is because the non-binary framework makes use of the costly cosine similarity rather than the hardware-friendly Hamming distance metric. In [35], BinHD uses three main blocks, encoding, associative search and counter modules, dealing with binary hypervectors. Their evaluation shows that, over four practical applications, the proposed BinHD can reach 12.4× and 6.3× energy efficiency and speedup in training process, while 13.8× and 9.9× during inference, compared to the state-of-art HD computing algorithm with a comparable classification accuracy.

7) HD Computing for Semi-Supervised Learning

In [53], SemiHD has been proposed as a self-training or self-learning approach for semi-supervised learning, where the training data is composed of a small portion of labeled data and a large portion of unlabeled data.

The SemiHD framework is depicted in Fig. 13 and the flow is illustrated as follows. 1). Encode all the data points, labeled and unlabeled, into HD space with $d = 10{,}000$ dimensions. 2). Start training from the labeled data to generate k hypervectors, each representing one class. 3). Predict the label for unlabeled data points. Labeling is performed by checking the similarity of unlabeled data with all the class hypervectors, and return the label which shows the highest similarity. 4). Select and add S% of unlabeled data with highest confidence to labeled data, where S is defined as the expansion rate. In [53], typically $S = 5$. 5). Redo the training task based on the expanded labeled data. Such iterative process stops when the accuracy does not change more than 0.1%. 6). Once the model has already been trained, perform the inference task by comparing the similarity of each test data with the trained model, to return the label with maximum similarity.

Their evaluation shows that the SemiHD can on average improve the classification of supervised HD by 10.2%. Additionally, compared to the best CPU implementation, the FPGA counterpart of SemiHD offers 7.11× faster speed and 12.6× energy efficiency.

8) HD Computing for Unsupervised Learning

HD computing has also been used in several unsupervised applications. See [57]–[61].

C. Images

1) HD Classification for Character Recognition

HD classification has been used for character recognition in [62] and later in [56]. As shown in Fig. 14, the input image is composed of 7 × 5 = 35 pixels. Each pixel has two possible values, that is 0 or 1, representing black or white. 1). Encode each pixel to a binary hypervector (indexHV). Totally 35 orthogonal indexHVs are stored in the item memory. 2). Based on HoloGN encoding—an encoding method proposed in [62] to address image data using HD computing—the indexHV is shifted depending on the pixel value. Accumulate all 35 indexHVs and perform a majority rule by a thresholding block to generate a holoHV for one input image. 3). The supervised controller will only be activated when this HD system conducts supervised learning. Otherwise, the system conducts the one-shot learning. The supervised controller accumulates the holoHVs for the same class and employs the thresholding block and generates the letterHV to be stored in the associative memory. The total number of letterHVs is 26. 4). During the test phase, the query hypervector is generated following the same module with test data. Then the similarity of each query hypervector is computed for all trained letterHVs to find the most similar class.

Results in [56] show that HD computing performs well for character recognition. Further optimization for HD computing may be conducted by reducing the dimensionality and increasing the input image size. The results also show that HD computing offers great robustness against noise. The system of 4,000-bit hypervectors achieves comparable average accuracy to its 12,000-bit counterpart at 0% distortion, and achieves an average accuracy of 89.94% with 14.29% distortion.

D. Summary

As mentioned above, HD computing shows great potential in dealing with data in the form of signals [28, 43, 65, 66, 75], letters [30, 64], and images [9, 56, 62], as long as these can be transformed into the HD space. Such preprocessing may include feature extraction and encoding. Evaluation shows that HD computing achieves good results for seizure detection [43, 66]. In addition, HD computing can also be combined with quantization technique to binarize HD model with minimal accuracy loss [76]. Table 3 offers more details about improvement strategies adopted in HD computing for accuracy and efficiency. As can been seen from Table 4, HD computing offers an acceptable accuracy, but with quite high efficiency. In some applications like DNA sequencing [31], HD computing outperforms other machine learning methods.

There still exist some interesting papers not discussed in detail in this review paper. Interested readers can refer to the following references, which include but are not limited to: 1). Considering the security issue when IoT devices release the offload computation to the cloud, [77] illustrates how the proposed SecureHD accelerates efficiency with high security. 2). To balance the tradeoff between efficiency and accuracy, QubitHD [76] is proposed as a stochastic binarization algorithm to achieve comparable accuracy to the non-binarized counterparts. SparseHD [37] takes advantage of the sparsity of the trained HD model for acceleration.

HD computing is still in its infancy. Future directions may include but is not limited to:

- More cognitive tasks: Inspired by [32], apart from the engineering aspect of HD computing, which is to solve classification tasks, more “cognition” aspects of HD computing should be explored. Such tasks include but are not limited to analogical reasoning, semantic generalization and relational representation.
- Feature exaction and encoding method: Since HD computing cannot directly address data like signals and images, feature exaction is vital to representation of information. For example, [75]

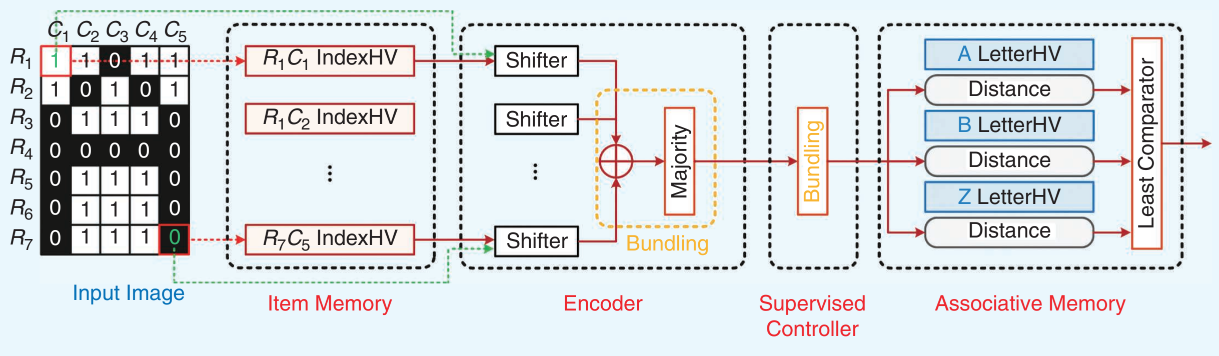

Figure 14. Block diagram of the HD character recognition system [56].

Table 3.
Summary of the strategies used in HD computing for accuracy and efficiency improvement.

		Model Type**							
Applications	Encode*	base/ level	train	test	Platform△	Accuracy♡	Acceleration♣	Motivation	Application
QuantHD [25]	1	B/B	B/T	B/T	F, C, G	Re, shuffle	Q, DR, F	speedup + accuracy	speech, activity, face, phone position
VoiceHD [28]	1	B/B	B	B	C	NN, Re	DR, B	Replace deep learning	speech
CompHD [51]	3	P	N	N	F	N	DR, Comp	DR without accuracy loss	activity, gesture, valve monitoring
AdaptHD [54]	1	B/B	N	B	C	Re, N	B, Adapt	accuracy + short time Re	speech, face, activity, Cardiotocograms
BinHD [35]	1	B/B	B	B	C	Re	B	speedup	speech, face, activity, Cardiotocograms
SemiHD [53]	1	B/B	B	B	F, C	Re, N	DR, B	Replace deep learning	17 popular datasets [63]
Language [21]	2	B	B	B	F	\	B	energy saving + robustness	language recognition
Character [56]	3	B	B	B	\	Binary	DR	data classification in IoT	character recognition
Laelap [43]	1	B	B	B	C, G	\	\	energy efficiency	seizure detection

* three encoding methods. 1: record-based encoding, 2: N-gram-based encoding, 3: a novel method.
** symbol "/" is used in record-based encoding. B: binary, P: bipolar, T: Ternary, N: non-binary.
△ implementation platforms. F: FPGA, C: CPU, G: GPU.
♡ strategies for accuracy improvement. Re: retraining, N: non-binary model, NN: neural network.
♣ strategies for efficiency improvement. DR: dimension reduction, Q: quantization, B: binarization, F: FPGA, Comp: compression, Adapt: adaptive.

Table 4.
Partial list of applications based on HD computing♣ in [40].

Applications	Inputs (#)*	Classes (#)**	HD (%)	Baseline (%)
Language recognition [21, 30]	1	21	96.70%	97.90%
Text categorization [64]	1	8	94.20%	86.40%
Speech recognition [28]	1	26	95.30%	93.60%
EMG gesture recognition [23]	4	5	97.80%	89.70%
Flexible EMG gesture recognition [27]	64	5	96.60%	88.90%
EEG brain-machine interface [65]	64	2	74.50%	69.50%
ECoG seizure detection [66]	100	2	95.40%	94.30%
DNA sequencing [31]	1	\	99.74%	94.53%
Character recognition [56]	1	10	89.94%	\

* represents the number of input data.
** represents the number of class hypervectors to be trained and stored in the associative memory.
♣ Other works, like [67, 68, 69, 70, 71, 72, 73, 74], are not listed in this table..

partially deals with this by addressing the problem of mapping data to a high-dimensional space.

- Similarity measurement: Though cosine similarity and Hamming distance are currently widely used, new metrics should be developed that are hardware-friendly and can lead to high accuracy.
- Multiple class hypervectors: Traditional classifiers use multi-dimensional features to train a classifier. Often ranking can be used to select a small number of features out of many features [78]. It is possible that multiple class hypervectors, similar to multiple features in traditional classification, can be generated to represent a class in HD classification. Subsequently, multiple query hypervectors will need to be compared with their corresponding class hypervectors for each class. This is a topic for further research.
- Accuracy improvement: Strategies like retraining should be explored to further improve the accuracy of HD computing.
- Hardware acceleration: Rebuilding the specific implementation for HD computing to store and manipulate a large amount of hypervectors may result in high speed and energy efficiency. Moreover, inspired by [32], which discusses tradeoffs related to the density of hypervectors, a choice between dense and sparse approaches should be accordingly made based on the application scenarios. For example, adopting sparse representation requires lower memory footprints.
- General HD computing processor: Inspired by [13], addressing different types of data with only one general processor containing a large word-length ALU is of great interest.
- Hybrid systems: Hybrid systems are partially based on HD computing and partially on conventional machine learning. Only a few examples exist so far [79]–[82]. Further research on this topic can be explored in future.

V. Conclusion

This paper has summarized the fundamental arithmetic operations for the emerging computing model of HD computing that might achieve high robustness, fast learning ability, hardware-friendly implementation, and energy efficiency. Mathematically, HD computing can be viewed as an alternative in dealing with machine learning problems. Though in its infancy, HD computing shows its potential to be used as a light-weight classifier for applications with limited resources. This model can achieve outstanding classification performance for certain problems like DNA sequencing. Balancing the tradeoff between accuracy and efficiency is an important area of research. Improvements include but are not limited to encoding, retraining, non-binary model and hardware acceleration. HD computing sometimes leads to outstanding classification accuracy, while sometimes achieves acceptable accuracy but high efficiency. Thus, users need to evaluate whether HD computing is suitable for their application. Additionally, HD computing can be used in applications such as seizure detection, speech recognition, character recognition and language detection. More "cognition" aspects of HD computing, including analogical reasoning, relationship representation and analysis, will need to be further developed in the future.

Acknowledgment

This paper has been supported in parts by the NSF under grant number CCF-1814759 and by the Chinese Scholarship Council (CSC). The authors thank all four reviewers for their numerous constructive comments and suggestions.

Lulu Ge received the B.S. degree from Nanjing University of Posts and Telecommunications (NJUPT), Nanjing, China, in 2015, the M.S. degree from the Southeast University, Nanjing, China, in 2018. She is currently pursuing the Ph.D. degree of electrical engineering at University of Minnesota, Minneapolis, MN, USA.

Keshab K. Parhi received the B.Tech. degree from the Indian Institute of Technology (IIT), Kharagpur, in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1984, and the Ph.D. degree from the University of California, Berkeley, in 1988. He has been with the University of Minnesota, Minneapolis, since 1988, where he is currently Distinguished McKnight University Professor and Edgar F. Johnson Professor of Electronic Communication in the Department of Electrical and Computer Engineering. He has published over 650 papers, is the inventor of 31 patents, and has authored the textbook VLSI Digital Signal Processing Systems (Wiley, 1999). His current research addresses VLSI architecture design of machine learning systems, hardware security, data-driven neuroscience and molecular/DNA computing. Dr. Parhi is the recipient of numerous awards including the 2017 Mac Van Valkenburg award and the 2012 Charles A. Desoer Technical Achievement award from the IEEE Circuits and Systems Society, the 2003 IEEE Kiyo Tomiyasu Technical Field Award, and a Golden Jubilee medal from the IEEE Circuits and Systems Society in 2000. He served as the Editor-in-Chief of the IEEE Trans. Circuits and Systems, Part-I during 2004 and 2005. He was elected a Fellow of the IEEE in 1996 and a Fellow of the American Association for the Advancement of Science (AAAS) in 2017.

References

[1] P. Kanerva, *Sparse Distributed Memory*. MIT press, 1988.

[2] P. Smolensky, "Tensor product variable binding and the representation of symbolic structures in connectionist systems," *Artif. Intell.*, vol. 46, no. 1–2, pp. 159–216, 1990. doi: 10.1016/0004-3702(90)90007-M.

[3] T. A. Plate, "Holographic reduced representations," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 623–641, 1995. doi: 10.1109/72.377968.

[4] P. Kanerva et al., "Fully distributed representation," *PAT*, vol. 1, no. 5, p. 10,000, 1997.

[5] D. A. Rachkovskij and E. M. Kussul, "Binding and normalization of binary sparse distributed representations by context-dependent thinning," *Neural Comput.*, vol. 13, no. 2, pp. 411–452, 2001. doi: 10.1162/089976601300014592.

[6] R. W. Gayler, "Multiplicative binding, representation operators & analogy (workshop poster)," 1998.

[7] K. Schlegel, P. Neubert, and P. Protzel, A comparison of vector symbolic architectures. 2020. [Online] Available: arXiv:2001.11797

[8] M. Hersche, J. d R. Millán, L. Benini, and A. Rahimi, Exploring embedding methods in binary hyperdimensional computing: A case study for motor-imagery based brain-computer interfaces. 2018. arXiv:1812.05705

[9] A. Rahimi et al., "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017. doi: 10.1109/TCSI.2017.2705051.

[10] R. E. Bryant and D. R. O'Hallaron, "Computer systems: A programmer's perspective," 2015.

[11] A. Patyk-Łońska, M. Czachor, and D. Aerts, "A comparison of geometric analogues of holographic reduced representations, original holographic reduced representations and binary spatter codes," in *Proc. 2011 Federated Conf. Computer Science and Information Systems (FedCSIS)*, IEEE, 2011, pp. 221–228.

[12] P. J. Olver and C. Shakiban, *Applied Linear Algebra*. Springer, 2018.

[13] S. Datta, R. A. Antonio, A. R. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric IoT," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 3, pp. 439–452, 2019. doi: 10.1109/JETCAS.2019.2935464.

[14] D. Widdows and T. Cohen, "Reasoning with vectors: A continuous model for fast robust inference," *Logic J. IGPL*, vol. 23, no. 2, pp. 141–173, 2015. doi: 10.1093/jigpal/jzu028.

[15] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–25, 2019. doi: 10.1145/3314326.

[16] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cogn. Comput.*, vol. 1, no. 2, pp. 139–159, 2009. doi: 10.1007/s12559-009-9009-8.

[17] D. Rachkovskij, "Linear classifiers based on binary distributed representations," *Int. J. Inform. Theories Appl.*, 2007.

[18] E. M. Kussul, L. M. Kasatkina, D. A. Rachkovskij, and D. C. Wunsch, "Application of random threshold neural networks for diagnostics of micro machine tool condition," in *Proc. 1998 IEEE Int. Joint Conf. Neural Networks. IEEE World Congress Computational Intelligence (Cat. No. 98CH36227)*, vol. 1. IEEE, 1998, pp. 241–244. doi: 10.1109/IJCNN.1998.682270.

[19] E. Kussul, "On image texture recognition by associative-projective neurocomputer," in *Proc. ANNIE'91 Conf, Intelligent Engineering Systems through Artificial Neural Networks*, ASME Press, 1991, pp. 453–458.

[20] D. Rachkovskij and T. Fedoseyeva, "On audio signals recognition by multilevel neural network," in *Proc. Int. Symp. Neural Networks and Neural Computing*, vol. 90, 1990, pp. 281–283.

[21] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proc. 2016 Int. Symp. Low Power Electronics and Design*, pp. 64–69. doi: 10.1145/2934583.2934624.

[22] B. Logan et al., "Mel frequency cepstral coefficients for music modeling," *ISMIR*, vol. 270, pp. 1–11, 2000.

[23] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition," in *Proc. 2016 IEEE Int. Conf. Rebooting Computing*, pp. 1–8. doi: 10.1109/ICRC.2016.7738683.

[24] D. Kleyko and E. Osipov, "Brain-like classifier of temporal patterns," in *Proc. 2014 Int. Conf. Computer and Information Sciences*, pp. 1–6. doi: 10.1109/ICCOINS.2014.6868349.

[25] M. Imani et al., "QuantHD: A quantization framework for hyperdimensional computing," in *Proc. IEEE Trans.Computer-Aided Design of Integrated Circuits and Systems*, 2019. doi: 10.1109/TCAD.2019.2954472.

[26] A. Rahimi, P. Kanerva, J. R. Millán, and J. M. Rabaey, "Hyperdimensional computing for noninvasive brain-computer interfaces: Blind and one-shot classification of EEG error-related potentials," in *Proc. 10th EAI Int. Conf. Bio-inspired Information and Communications Technologies*, 2017. doi: 10.4108/eai.22-3-2017.152397.

[27] A. Moin et al., "An EMG gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," in *Proc. 2018 IEEE Int. Symp. Circuits and Systems (ISCAS)*, pp. 1–5. doi: 10.1109/ISCAS.2018.8351613.

[28] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "VoiceHD: Hyperdimensional computing for efficient speech recognition," in *Proc. 2017 IEEE Int. Conf. Rebooting Computing (ICRC)*, pp. 1–8. doi: 10.1109/ICRC.2017.8123650.

[29] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *Proc. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6. doi: 10.1109/DAC.2018.8465708.

[30] A. Joshi, J. T. Halseth, and, and P. Kanerva, "Language geometry using random indexing," in *Proc. Int. Symp. Quantum Interaction*, Springer, 2016, pp. 265–274. doi: 10.1007/978-3-319-52289-0_21.

[31] M. Imani, T. Nassar, A. Rahimi, and T. Rosing, "HDNA: Energy-efficient DNA sequencing using hyperdimensional computing," in *Proc. 2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*. IEEE, 2018, pp. 271–274. doi: 10.1109/BHI.2018.8333421.

[32] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey, "Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 5880–5898, 2018. doi: 10.1109/TNNLS.2018.2814400.

[33] R. W. Gayler, Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. 2004. [Online]. Available: arXiv:cs/0412059

[34] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of ExG signals," *Proc. IEEE*, vol. 107, no. 1, pp. 123–143, 2018. doi: 10.1109/JPROC.2018.2871163.

[35] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *Proc. 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 126–131. doi: 10.23919/DATE.2019.8714821.

[36] D. A. Rachkovskij, "Representation and processing of structures with binary sparse distributed codes," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 2, pp. 261–276, 2001. doi: 10.1109/69.917565.

[37] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "SparseHD: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *Proc. 2019 IEEE 27th Annu. Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, pp. 190–198. doi: 10.1109/FCCM.2019.00034.

[38] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible FPGA-based framework for refreshing hyperdimensional computing," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pp. 53–62. doi: 10.1145/3289602.3293913.

[39] G. Karunaratne, M. L. Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, In-memory hyperdimensional computing. 2019. [Online]. Available: arXiv:abs/1906.01548

[40] A. Rahimi, T. F. Wu, H. Li, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, Hyperdimensional computing nanosystem. 2018. [Online]. Available: arXiv:1811.09557

[41] T. F. Wu et al., "Hyperdimensional computing exploiting carbon nanotube FETs, resistive RAM, and their monolithic 3D integration," *IEEE J. Solid-State Circuits*, vol. 53, no. 11, pp. 3183–3196, 2018. doi: 10.1109/JSSC.2018.2870560.

[42] H. Li et al., "Hyperdimensional computing with 3D VRRAM in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *Proc. 2016 IEEE Int. Electron Devices Meeting*, pp. 16–11. doi: 10.1109/IEDM.2016.7838428.

[43] A. Burrello, L. Cavigelli, K. Schindler, L. Benini, and A. Rahimi, "Laelaps: An energy-efficient seizure detection algorithm from long-term human iEEG recordings without false alarms," in *Proc. 2019 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 752–757. doi: 10.23919/DATE.2019.8715186.
[44] "UCI machine learning repository," [Online]. Available: http://archive.ics.uci.edu/ml/datasets/ISOLET
[45] Z. Zhang and K. K. Parhi, "Seizure detection using wavelet decomposition of the prediction error signal from a single channel of intra-cranial EEG," in *Proc. 2014 36th Annu. Int. Conf. IEEE Engineering Medicine and Biology Society*, pp. 4443–4446. doi: 10.1109/EMBC.2014.6944610.
[46] Z. Zhang and K. K. Parhi, "Low-complexity seizure prediction from iEEG/sEEG using spectral power and ratios of spectral power," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 3, pp. 693–706, 2015. doi: 10.1109/TBCAS.2015.2477264.
[47] Z. Zhang and K. K. Parhi, "Seizure detection using regression tree based feature selection and polynomial SVM classification," in *Proc. 2015 37th Annu. Int. Conf. IEEE Engineering Medicine and Biology Society*, pp. 6578–6581. doi: 10.1109/EMBC.2015.7319900.
[48] Z. Zhang and K. K. Parhi, "Seizure prediction using polynomial SVM classification," in *Proc. 2015 37th Annu. Int. Conf. IEEE Engineering Medicine and Biology Society (EMBC)*, pp. 5748–5751. doi: 10.1109/EMBC.2015.7319900.
[49] K. K. Parhi and Z. Zhang, "Discriminative ratio of spectral power and relative power features derived via frequency-domain model ratio with application to seizure prediction," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 4, pp. 645–657, 2019. doi: 10.1109/TBCAS.2019.2917184.
[50] S. I. Gallant and T. W. Okaywe, "Representing objects, relations, and sequences," *Neural Comput.*, vol. 25, no. 8, pp. 2038–2078, 2013. doi: 10.1162/NECO_a_00467.
[51] J. Morris, M. Imani, S. Bosch, A. Thomas, H. Shu, and T. Rosing, "CompHD: Efficient hyperdimensional computing using model compression," in *Proc. 2019 IEEE/ACM Int. Symp. Low Power Electronics and Design (ISLPED)*, pp. 1–6. doi: 10.1109/ISLPED.2019.8824908.
[52] "Hadamard matrix," [Online]. Available: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.linalg.hadamard.html
[53] M. Imani et al. "SemiHD: Semi-supervised learning using hyperdimensional computing," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, 2019, pp. 1–8, doi: 10.1109/ICCAD45719.2019.8942165.
[54] M. Imani, J. Morris, S. Bosch, H. Shu, G. D Micheli, and T. Rosing, "AdaptHD: Adaptive efficient training for brain-inspired hyperdimensional computing," in *Proc. 2019 IEEE Biomedical Circuits and Systems Conf.*, pp. 1–4. doi: 10.1109/BIOCAS.2019.8918974.
[55] S. S. Haykin, *Adaptive Filter Theory*. Pearson Education India, 2014.
[56] A. X. Manabat, C. R. Marcelo, A. L. Quinquito, and A. Alvarez, "Performance analysis of hyperdimensional computing for character recognition," in *Proc. 2019 Int. Symp. Multimedia and Communication Technology*, pp. 1–5. doi: 10.1109/ISMAC.2019.8836136.
[57] P. Kanerva, J. Kristoferson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proc. Annu. Meeting Cognitive Sci. Soc.*, vol. 22, 2000.
[58] G. Recchia, M. Sahlgren, P. Kanerva, and M. N. Jones, "Encoding sequential information in semantic space models: Comparing holographic reduced representation and random permutation," *Comput. Intell. Neurosci.*, vol. 2015, 2015. doi: 10.1155/2015/986574.
[59] D. Kleyko, E. Osipov, D. De Silva, U. Wiklund, V. Vyatkin, and D. Alahakoon, "Distributed representation of n-gram statistics for boosting self-organizing maps with hyperdimensional computing," in *Proc. Int. Andrei Ershov Memorial Conf. Perspectives System Informatics*, Springer, 2019, pp. 64–79. doi: 10.1007/978-3-030-37487-7_6.
[60] T. Bandaragoda, D. De Silva, D. Kleyko, E. Osipov, U. Wiklund, and D. Alahakoon, "Trajectory clustering of road traffic in urban environments using incremental machine learning in combination with hyperdimensional computing," in *Proc. 2019 IEEE Intelligent Transportation Systems Conf.*, pp. 1664–1670. doi: 10.1109/ITSC.2019.8917320.
[61] M. Hersche, S. Sangalli, L. Benini, and A. Rahimi, "Evolvable hyperdimensional computing: Unsupervised regeneration of associative memory to recover faulty components," in *Proc. IEEE Int. Conf. Artificial Intelligence Circuits and Systems (AICAS), Genoa, Italy*, 2020. doi: 10.1109/AICAS48895.2020.9073871.
[62] D. Kleyko, E. Osipov, A. Senior, A. I. Khan, and Y. A. Şekercioğğlu, "Holographic graph neuron: A bioinspired architecture for pattern processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 6, pp. 1250–1262, 2016. doi: 10.1109/TNNLS.2016.2535338.
[63] I. Triguero, S. García, and F. Herrera, "Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study," *Knowledge Inform. Syst.*, vol. 42, no. 2, pp. 245–284, 2015. doi: 10.1007/s10115-013-0706-y.
[64] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," in *Proc. Design, Automation Test in Europe Conf. Exhibition (DATE), University Booth*, 2016, p. 1.
[65] A. Rahimi, A. Tchouprina, P. Kanerva, J. d R. Millán, and J. M. Rabaey, "Hyperdimensional computing for blind and one-shot classification of EEG error-related potentials," *Mobile Netw. Appl.*, pp. 1–12, 2017. doi: 10.1007/s11036-017-0942-6.
[66] A. Burrello, K. Schindler, L. Benini, and A. Rahimi, "One-shot learning for iEEG seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *Proc. 2018 IEEE Biomedical Circuits and Systems Conf.*, pp. 1–4. doi: 10.1109/BIOCAS.2018.8584751.
[67] O. Räsänen and S. Kakouros, "Modeling dependencies in multiple parallel data streams with hyperdimensional computing," *IEEE Signal Process. Lett.*, vol. 21, no. 7, pp. 899–903, 2014. doi: 10.1109/LSP.2014.2320573.
[68] O. J. Räsänen and J. P. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 9, pp. 1878–1889, 2015. doi: 10.1109/TNNLS.2015.2462721.
[69] D. Kleyko, E. Osipov, R. W. Gayler, A. I. Khan, and A. G. Dyer, "Imitation of honey bees concept learning processes using vector symbolic architectures," *Biol. Inspired Cogn. Arch.*, vol. 14, pp. 57–72, 2015. doi: 10.1016/j.bica.2015.09.002.
[70] O. Yilmaz, Connectionist-symbolic machine intelligence using cellular automata based reservoir-hyperdimensional computing. 2015. [Online]. Available: arXiv:1503.00851
[71] D. Kleyko, S. Khan, E. Osipov, and S.-P. Yong, "Modality classification of medical images with distributed representations based on cellular automata reservoir computing," in *Proc. 2017 IEEE 14th Int. Symp. Biomedical Imaging*, pp. 1053–1056. doi: 10.1109/ISBI.2017.7950697.
[72] G. Montone, J. K. O'Regan, and A. V. Terekhov, Hyper-dimensional computing for a visual question-answering system that is trainable end-to-end. 2017. [Online]. Available: arXiv:1711.10185
[73] D. Kleyko, E. Osipov, N. Papakonstantinou, and V. Vyatkin, "Hyperdimensional computing in industrial systems: the use-case of distributed fault isolation in a power plant," *IEEE Access*, vol. 6, pp. 30,766–30,777, 2018. doi: 10.1109/ACCESS.2018.2840128.
[74] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Sci. Robot.*, vol. 4, no. 30, p. eaaw6736, 2019. doi: 10.1126/scirobotics.aaw6736.
[75] O. J. Räsänen, "Generating hyperdimensional distributed representations from continuous-valued multivariate sensory input," in *Proc. CogSci*, 2015.
[76] S. Bosch, A. S. de la Cerda, M. Imani, T. S. Rosing, and G. De Micheli, QubitHD: A stochastic acceleration method for HD computing-based machine learning. 2019. [Online] Available: arXiv:1911.12446
[77] M. Imani et al., "A framework for collaborative learning in secure high-dimensional space," in *Proc. 2019 IEEE 12th Int. Conf. Cloud Computing*, pp. 435–446. doi: 10.1109/CLOUD.2019.00076.
[78] Z. Zhang and K. K. Parhi, "MUSE: Minimum uncertainty and sample elimination based binary feature selection," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 9, pp. 1750–1764, 2018. doi: 10.1109/TKDE.2018.2865778.
[79] P. Alonso, K. Shridhar, D. Kleyko, E. Osipov, and M. Liwicki, HyperEmbed: Tradeoffs between resources and performance in NLP tasks with hyperdimensional computing enabled embedding of n-gram statistics. 2020. [Online]. Available: arXiv:2003.01821
[80] D. Kleyko, M. Kheffache, E. P. Frady, U. Wiklund, and E. Osipov, Density encoding enables resource-efficient randomly connected neural networks. 2019. [Online]. Available: arXiv:1909.09153
[81] D. Kleyko, E. P. Frady, and E. Osipov, Integer echo state networks: Hyperdimensional reservoir computing. 2017. [Online]. Available: arXiv:1706.00280
[82] A. G. Anderson and C. P. Berg, The high-dimensional geometry of binary neural networks. 2017. [Online]. Available: arXiv:1705.07199