BRIC: Locality-based Encoding for Energy-Efficient Brain-Inspired Hyperdimensional Computing

Mohsen Imani, Justin Morris, John Messerly, Helen Shu, Yaobang Deng, and Tajana Rosing
CSE Department, UC San Diego, La Jolla, CA 92093, USA
{moimani, j1morris, jmesserl, hsshu, yad025, tajana}@ucsd.edu

ABSTRACT

Brain-inspired Hyperdimensional (HD) computing is a new computing paradigm emulating the neuron's activity in high-dimensional space. The first step in HD computing is to map each data point into high-dimensional space (e.g., 10,000), which requires the computation of thousands of operations for each element of data in the original domain. Encoding alone takes about 80% of the execution time of training. In this paper, we propose BRIC, a fully binary Brain-Inspired Classifier based on HD computing for energy-efficient and high-accuracy classification. BRIC introduces a novel encoding module based on random projection with a predictable memory access pattern which can efficiently be implemented in hardware. BRIC is the first HD-based approach which provides data projection with a 1:1 ratio to the original data and enables all training/inference computation to be performed using binary hypervectors. To further improve BRIC efficiency, we develop an online dimension reduction approach which removes *insignificant* hypervector dimensions during training. Additionally, we designed a fully pipelined FPGA implementation which accelerates BRIC in both training and inference phases. Our evaluation of BRIC a wide range of classification applications show that BRIC can achieve 64.1× and 9.8× (43.8× and 6.1×) energy efficiency and speed up as compared to baseline HD computing during training (inference) while providing the same classification accuracy.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**; *Supervised learning*;

KEYWORDS

Brain-inspired computing, Hyperdimensional computing, Machine learning, Energy efficiency

ACM Reference Format:
Mohsen Imani, Justin Morris, John Messerly, Helen Shu, Yaobang Deng, and Tajana Rosing. 2019. BRIC: Locality-based Encoding for Energy-Efficient Brain-Inspired Hyperdimensional Computing. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317785

1 INTRODUCTION

The emergence of the Internet of Things (IoT) has led to a copious amount of small connected embedded devices. Many of these devices need to perform classification tasks such as speech recognition, activity recognition, face detection, and medical diagnosis [1, 2]. However, these small embedded devices do not have the computing power to run sophisticated classification algorithms such as Deep Neural Networks(DNN) [3]. To resolve this, many devices send the data they collect to the cloud and the cloud performs the inference task, sending the result back to the embedded device. This leads to new problems such as network usage and user security [4]. In order to solve these new issues and still provide a way for these embedded devices to perform classification tasks, we need a light-weight classification algorithm that can achieve comparable accuracy to sophisticated resource-intensive algorithms.

Brain-inspired Hyperdimensional (HD) computing has been proposed as the alternative computing method that processes the cognitive tasks in a more light-weight way [5]. HD computing is developed based on the fact that brains compute with *patterns of neural activity* [5]. Recent research utilized high dimension vectors (e.g., more than a thousand dimension), called *hypervectors*, to represent the neural activities, and showed successful progress for many cognitive tasks such as activity recognition, object recognition, language recognition, and bio-signal classification [6–8]. HD computing offers an efficient learning strategy without overcomplex computation steps such as back propagation in neural networks. In addition, it builds upon a well-defined set of operations with random HD vectors which makes the learning model extremely robust in the possible presence of hardware failures.

In HD computing, training data points are combined into a set of hypervectors, called *an HD model*, through light-weight computation steps. Each hypervector in the model represents a class of the target classification problem. Most of the proposed HD computing work exploit binarized hypervectors to reduce the computational/memory intensity in HD computing [9, 10]. However, the existing HD computing algorithms [9] have two main challenges: (i) the encoding is computationally expensive, as it requires the computation of thousands (e.g., 10,000) of operations to map each element of data from the original domain to high-dimensional space [8, 11]. For example, our experiments on five practical applications (described in Section 6.1) show that in HD computing the encoding module takes about 79% and 74% of the training and inference time. (ii) In addition, HD computing using binary encoded vectors provides significantly lower classification accuracy. In other words, HD computing requires non-binary (integer) vectors in order to provide acceptable accuracy. However, working with non-binary vectors significantly increases the memory requirement and the computation complexity of training and inference.

In this paper, we propose BRIC, a fully binary Brain-Inspired Classifier based on HD computing for energy-efficient and high-accuracy classification. BRIC introduces a novel encoding module based on random projection with a predictable memory access pattern which can be efficiently implemented in hardware. In contrast to existing HD computing algorithms that increase the size of encoded data by 20× [9], BRIC is the first HD-based approach which provides data projection with a 1:1 ratio to the original data. In addition, BRIC enables all training/inference computation to be performed using binary hypervectors. The low memory requirement and computation cost makes BRIC a suitable candidate for

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317785

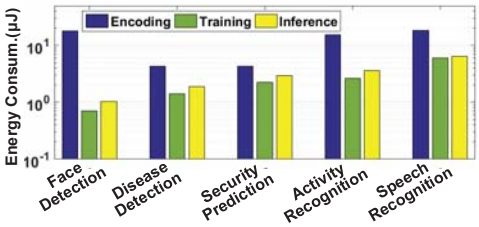

Figure 1: Energy consumption of HD encoding, training, and inference.

embedded devices with limited resources. To further improve BRIC efficiency, we developed an online dimension reduction approach which removes *insignificant* hypervector dimensions during training. Additionally, we designed a fully pipelined FPGA implementation which accelerates BRIC in both training and inference phases. Our evaluation on five practical classification applications, shows that BRIC can achieve 64.1× and 9.8× (43.8× and 6.1×) energy efficiency and speed up as compared to baseline HD during training (inference) while providing the same classification accuracy.

2 RELATED WORK & MOTIVATION

Prior work tried to apply the idea of high-dimensional computing to different classification problems such as language recognition, speech recognition, face detection, EMG gesture detection, human-computer interaction, and sensor fusion prediction [6, 8–10, 12–14]. For example, work in [11] proposed a simple and scalable alternative to latent semantic analysis. Additionally, work in [8] proposed a new HD encoding based on random indexing for recognizing a text's language by generating and comparing text hypervectors. Work in [15] proposed an encoding method to map and classify biosignal sensory data in high dimensional space. Work in [7, 9] proposed a general encoding module that maps feature vectors into high-dimensional space while keeping most of the original data.

Prior work also tried to design hardware acceleration for HD computing by mapping its operations into hardware, e.g., in-memory architecture [16–21], and tried to accelerate HD computing in hardware by binarizing the class hypervectors [22] or removing dimensions of the class hypervectors [23]. Work in [24] designed an FPGA implementation to accelerate HD computation in the binary domain. However, the application of these approaches is limited to simple classification problems such as language recognition [8]. In order to provide acceptable classification accuracy, all these approaches have to train the model using non-binary (integer) vectors. However, using non-binary vectors requires a large memory footprint and computation cost in both training and inference.

In this work, we observe that the existing encoding modules are algorithmically and computationally inefficient. In addition, in order to get high accuracy, the encoding needs to map data into vectors with integer values which significantly increases the data size [9, 10]. This large size memory is not often available on embedded devices with limited resources. Figure 1 shows the energy consumption of encoding, training, and inference (associative search) when running a single data point on five practical applications. Our evaluation shows that the encoding module on average takes 4.7× and 3.8× higher energy than HD training and inference. In this work, we propose a novel encoding approach that (i) significantly reduces the encoding computation cost by introducing computation locality and (ii) provides high classification accuracy while mapping data into binary vectors with much lower dimensionality than existing algorithms. Additionally, training can be performed using encoded vectors in the binary domain. Our approach also simplifies the inference similarity check to *Hamming distance* with minimal quality loss as compared to the existing HD computing algorithm with an integer model. We also propose a fully pipelined FPGA implementation that accelerates a wide range of classification problems during training and inference.

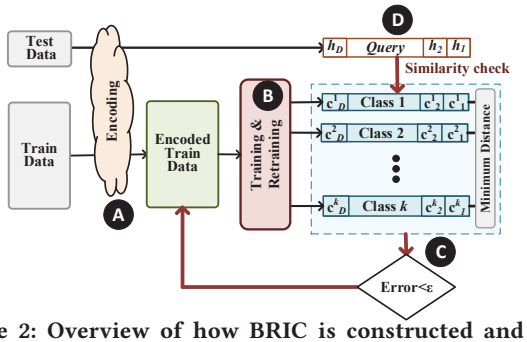

Figure 2: Overview of how BRIC is constructed and how BRIC performs inference.

3 PROPOSED BRIC

In this paper, we propose BRIC, a novel framework for fully binary classification. BRIC consists of three main modules shown in Figure 2: encoding, training, and testing. The encoding module maps each data point to binary high-dimensional space. Our encoding has been designed to map the maximum amount of information to high dimensional space with the minimum computation cost. Binarizing the model enables BRIC inference to be supported using a low-cost Hamming distance similarity check. In the following, we explain the details of BRIC functionality.

3.1 Encoding

Random Projection: Figure 2A shows the overview of BRIC performing the classification task. Before we can work in high dimension space, we first need to encode the data to hypervectors. We desire a fast and hardware-friendly algorithm that can take a vector of real-valued data and generate a binary code such that the encoding preserves the cosine similarity. Let us assume $\mathbf{A}, \mathbf{B} \in \mathbb{R}^n$ are two feature vectors in the original domain with real values. We wish to define an encoding operation $\lambda(*)$ such that:

$$\{\mathbf{X} = \lambda(\mathbf{A}), \mathbf{Y} = \lambda(\mathbf{B}) \,,\ \mathbf{X}, \mathbf{Y} \in \{1, -1\}^D\}$$

$$\delta(\mathbf{A}, \mathbf{B}) = \delta(\mathbf{X}, \mathbf{Y})$$

where $\delta(*)$ is the cosine similarity. Since the cosine angle of binary vectors is determined by how many bits match, the cosine angle and Hamming distance are proportional. This type of encoding can be performed using Locality Sensitive Hash algorithms, such as Random Projection [25]. Let us assume a feature vector $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$, with n features ($f_i \in \mathbb{N}$) in original domain. The goal of random projection is to map this feature vector to a D dimensional space vector: $\mathbf{H} = \{h_1, h_2, \ldots, h_D\}$. As Figure 3a shows, random projection generates D dense bipolar vectors with the same dimensionality as original domain, $\{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_D\}$, where $\mathbf{P}_i \in \{-1, 1\}^n$. The inner product of a feature vector with each randomly generated vector gives us a single dimension of a hypervector in high-dimensional space. For example, we can compute the $i-th$ dimension of the encoded data as:

$$h_i = sign(\mathbf{P}_i \cdot \mathbf{F})$$

where *sign* is a sign function which maps the result of the dot product to +1 or -1 values. This type of hashing involves a large amount of multiplications/additions which is inefficient in hardware. For example, to map a feature vector from n to D dimensions, this encoding involves $n \times D$ multiplication and addition operations.

Sparse Random Encoding: The efficiency of random projection can be improved by sparsifying each projection vector. Instead of generating dense projection vectors, we can generate sparse projection vectors(Figure 3b). Consider s as a sparsity of each projection vector. Then, for each sparse projection vector, only $s\%$ of the vector's elements are randomly generated and the rest are set to zero. For example, if $s = 5\%$, each projection vector only has $0.05 \times n$ non-zero elements. Therefore, each dimension of the encoded hypervector can be computed with only $0.05 \times n$ multiplication/addition operations. Therefore, encoding a single hypervector takes $s \times n \times D$ multiplication/addition operations, compared to $n \times D$ multiplication/addition operations with dense projection vectors. Although the sparsity significantly reduces the number of arithmetic operations, it introduces random accesses to the algorithm, which is hard on the cache and slows down the computation.

Locality-based Sparse Random Projection: Here we propose a novel approach that keeps the advantages of a sparse projection matrix, i.e., fewer operations, while removing random accesses to make the algorithm more hardware friendly. We propose a locality-based random projection encoding that uses a predictable access pattern. Instead of selecting s% random indices of the projection matrix to be non-zero, we approximate sparse random projection by selecting pre-defined indices to be non-zero. Figure 3c shows the structure of the locality-based matrix. Our approach selects the first $s \times n$ of the $\mathbf{P}_1$ vector to be non-zero (indices $[1...s \times n]$). Similarly, P_2 projection vector only has $s \times n$ non-zero elements on indices $[2...s \times n - 1]$. Finally, $\mathbf{P}_D$ contains non-zero elements on the last $s \times n$ dimensions. This creates a clear spacial locality pattern that hardware accelerators can take advantage of.

Figure 4 shows the overview of BRIC encoding mapping each n dimensional feature vector to a D dimensional binary hypervector. BRIC simplifies the projection matrix to a single dense random projection vector with D bipolar values. Our approach first replicates the feature vector, $\mathbf{F}$, such that it extends to D dimensions, the same as our desired high-dimensional vector. For example, to encode a feature vector with $n = 500$ features to $D = 4,000$ dimensions, we need to concatenate 8 copies of a feature vector together. Then, it generates a random D dimensional projection vector, P, next to the extended feature vector (as shown in Figure 4). To compute the dimensions of the high-dimensional vector, BRIC takes the dot product of the extended feature vector with each projection vector in an N-gram window. The first N-gram calculates the dot product of the first N features and N projection vector elements:

$$h_1 = sign(f_1 * p_1 + f_2 * p_2 + ... + f_N * p_N)$$

Similarly, the N-gram window shifts by a single position to generate the next feature values. So, we can compute the i^{th} dimension of an encoded hypervector using:

$$h_i = sign(f_i * p_i + f_{i+1} * p_{i+1} + ... + f_{i+N} * p_{i+N})$$

Each step of the N-gram window corresponds to a multiplication with a sparse projection vector in the projection matrix. Although this encoding has the same number of computations as sparse random projection, it provides the following advantages: (i) it removes random accesses from the feature selection by introducing spacial locality, which significantly reduces the cost of hardware implementation. (ii) There is an opportunity for computation reuse, as every neighboring dimension shares $N - 1$ terms.

3.2 BRIC Training

Initial Training: Figure 2 Ⓑ shows the functionality of BRIC during training. In HD computing, training is performed by element-wise addition of all encoded hypervectors in each existing class.

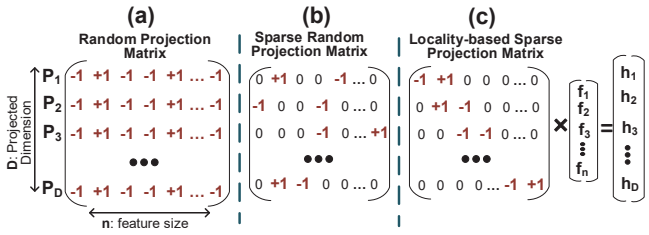

Figure 3: Random projection encoding using dense, sparse, and locality-based projection matrix.

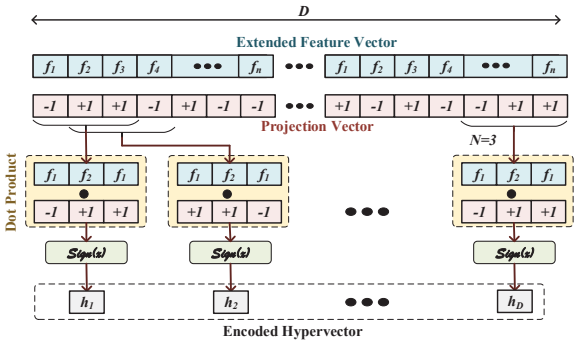

Figure 4: Locality-based random projection encoding.

The result of training are k hypervectors with D dimensions, where k is the number of classes. For example, the i^{th} class hypervector can be computed as: $\mathbf{C_i} = \sum_{\forall j \in class_i} \mathbf{H_j}$. This training operation involves a large amount of integer additions, which makes the HD computation costly [9]. Since our encoded hypervectors are bipolar, this accumulation can be performed more efficiently with binary accumulations. In BRIC, the resulting class hypervectors are also binarized, creating a binary model, where the class hypervector elements are changed to +1 if they are positive and -1 if they are negative or 0. The binary model is used for more efficient inference, however, the accumulated hypervectors are still stored as an integer model because they will be used for retraining.

Retraining: HD computing performs model adjustment by iteratively going through the training dataset. Figure 2 Ⓑ shows the functionality of BRIC during retraining. During a single iteration of model adjustment, HD computing checks the similarity of all training data points, say $\mathbf{H}$, with the trained binary model. If a data is wrongly classified by the model, HD updates the model by (i) adding the incorrectly classified hypervector to the class that input belongs to ($\widetilde{\mathbf{C}}^{correct} = \mathbf{C}^{correct} + \mathbf{H}$), and (ii) subtracting it from the class which it is wrongly matched with ($\widetilde{\mathbf{C}}^{wrong} = \mathbf{C}^{wrong} - \mathbf{H}$). These updates are done to the integer model saved from training because adding to and subtracting from the binary model would drastically change the model. To update the binary model, the updated class hypervectors from the integer model are sent to the bipolar domain using the same *Sign* function used for training. After each retraining iteration, we check the classification accuracy in the last three iterations and decide to stop the retraining if the change in error is less than 0.1% (Figure 2 Ⓒ). The retraining stops after 20 iterations if the convergence condition is not satisfied.

3.3 BRIC Inference

After training and retraining, the HD model can now be used for inference (Figure 2 Ⓓ). Upon inference, an input data is encoded to a *query* hypervector using the same encoding module used for training. HD computing then compares the similarity of the query

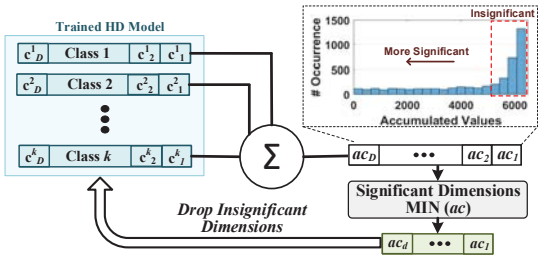

Figure 5: Online dimension reduction after initial training.

hypervector with all stored class hypervectors and selects the class with the highest similarity. Since BRIC generates a binary model, it uses Hamming distance metrics to find a class hypervector with the most similarity to the query hypervector. Comparatively, the existing HD computing algorithms [9] with integer models have to use *Cosine* similarity for inference. This is significantly more computationally expensive than Hamming distance, especially when implemented with an FPGA.

4 ONLINE DIMENSION REDUCTION

Online feature reduction attempts to remove insignificant "noisy" dimensions from the model to improve the efficiency of BRIC. In order to identify these dimensions, BRIC uses the distribution of the encoded training hypervectors. Figure 5 shows the distribution of the training hypervector values. This data is gathered by adding all the class hypervectors together after initial training.

The dimensions in which the data has high variation are closer to 0 because a high variation implies an equal amount of +1 and -1 accumulations. The dimensions in which the data has low variation will be farther than 0 because, the additions accumulate more sequential 1's or -1's. We declare the dimensions close to 0 to be "significant", while the dimensions farther away from zero to be "insignificant". This is because to distinguish the classes from each other, we want to emphasize their differences and not their similarities. BRIC drops the $s\%$ most insignificant dimensions from the model, resulting in a efficiency improvement of approximately $s\%$.

5 FPGA ACCELERATION

BRIC can be accelerated on different platforms such as CPU, GPU, FPGA or ASIC. FPGA is one of the best options as BRIC computation involves bitwise operations among long vector sizes. General strategies of optimizing the performance of BRIC are (i) using a pipeline and partial unrolling on low levels (dimension levels) to speed up each individual task and (ii) using dataflow design on a high level (task level) to build a stream processing architecture that lets different tasks run concurrently. In the following, we explain the functionality of BRIC in encoding, training, retraining, and inference phases.

5.1 Encoding Implementation

As we explained in Section 3.1, we used the locality-based random projection encoding to implement the encoding module. Due to the sequential and predictable memory access patterns as well as the abundance of binary operations, this encoding approach can be implemented efficiently on an FPGA. In the hardware implementation, we represent all $\{-1, +1\}$ values with $\{0, 1\}$ respectively. This enables us to represent each element of projection vector using a single bit. Figure 6a shows the hardware implementation of the BRIC encoding module. The encoding process includes reading a feature vector from off-chip DDR memory and generating a binary hypervector from them.

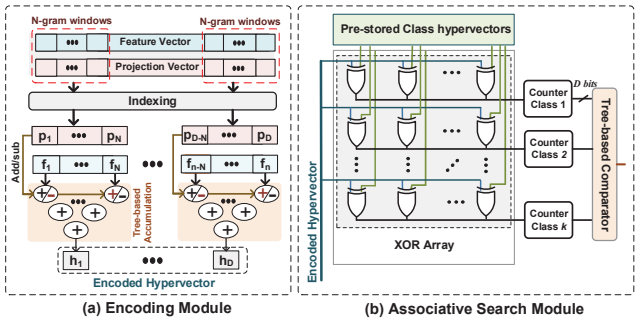

Figure 6: FPGA implementation of the encoding and associative search block.

Calculating the inner product of a feature vector and a projection vector, $P \in \{1, -1\}^D$, can be implemented with no multiplications. Each element of the projection vector decides the sign of each dimension of the feature vector in the accumulation of the dot product. Thus, the dot product can be simplified to addition/subtraction of the feature vector elements. Right after the encoding, the hypervectors are used for initial model training. We also need to store the encoded hypervectors for retraining. However, the FPGA does not have enough BRAM blocks to store all encoded hypervectors, so, our design stores them into DDR memory.

5.2 Training Implementation

Initial Training: BRIC initial training is a single-pass process. The training module accesses the encoded hypervectors and accumulates them in order to create a hypervector representing each class. When the training module accumulates the encoded hypervector to one of the class hypervectors, the encoding module maps the next training data into high-dimensional space, improving data throughput by increasing resource utilization. After going through all of the training data, our implementation binarizes the model by comparing each class hypervector with a threshold value. Finally, the binary model is stored in the BRAM blocks in order to be used for inference or retraining.

Retraining: The retraining phase first sequentially reads already encoded training hypervectors from the off-chip memory. Next, we check the similarity of each data point with all trained class hypervectors. Each data point gets a tag of a class which it has the highest Hamming distance similarity with. Figure 6b shows an overview of the implementation of the Hamming distance similarity between a query and class hypervectors. This similarity check is implemented using a XOR array which compares the bit similarity between two hypervectors. Counter blocks, shown in Figure 6b, calculate the number of mismatches of each class hypervector with the query data point. Finally, a tree-based comparator block finds the class with the highest counter value. In the case of a misclassification, BRIC needs to update the model by adding and subtracting a data hypervector with two class hypervectors as defined before.

5.3 Inference Implementation

After the retraining, BRIC has a stable model that can be used in the inference phase. The encoding module is integrated with the similarity check module as the entire inference part. Each test data point is first encoded to high-dimensional space using the same encoding block explained in Section 5.1. Next, BRIC checks the Hamming distance similarity of the data point with all prestored class hypervectors, in order to find a class with the highest similarity. One unique advantage of our approach is its capability to

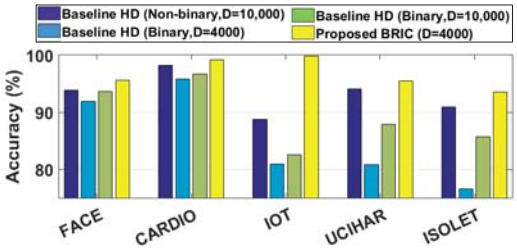

Figure 7: Classification accuracy of BRIC and the baseline HD using binary and integer models.

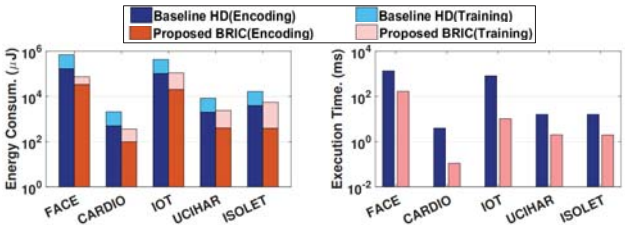

Figure 8: Energy consumption and execution time of BRIC and the baseline HD during training.

enable online training during inference phase. Our implementation stores two HD models: one with integer values used for retraining and a binary model which is used to perform the classification task. BRIC binarizes the integer model periodically in order to update the inference model.

6 EVALUATION

6.1 Experimental Setup

We implemented BRIC training, retraining, and inference in both software and hardware. In software, we implemented BRIC with Python. In hardware, we fully implemented BRIC using Verilog. We verify the timing and the functionality of the models by synthesizing them using Xilinx Vivado Design Suite [26]. The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit. We evaluated the efficiency of the proposed BRIC on four practical classification problems listed below: Speech Recognition (ISOLET) [27], Activity Recognition (UCIHAR) [28], Face Detection (FACE) [29], Cardiotocography (CARDIO) [30], and Attack Detection in IoT systems (IoT) [31].

6.2 BRIC Accuracy and Memory Requirement

Figure 7 compares the impact of hypervector dimensions on the classification accuracy of BRIC and the baseline HD computing algorithm [9]. As we explained, BRIC always encodes data points into D binary dimensions. However, for the baseline HD computing algorithm, we consider two cases when HD encodes data points to binary and integer domains. Our results in Figure 7 indicate that BRIC requires significantly fewer dimensions to provide the same accuracy as the baseline. For example, BRIC using $D = 4,000$ binary dimensions provides the same accuracy as the baseline with $D = 10,000$ integer dimensions. In addition, the baseline with a binarized model provides significantly lower accuracy than BRIC and the baseline with an integer model. BRIC is on average 11.5% more accurate than the baseline using a binary encoding and binary model.

Here we compare BRIC and the baseline in terms of the training memory requirement. As we explained in Section 3.2, the baseline/BRIC store all encoded training data in memory. Going into high dimensional space intuitively means increasing the data size,

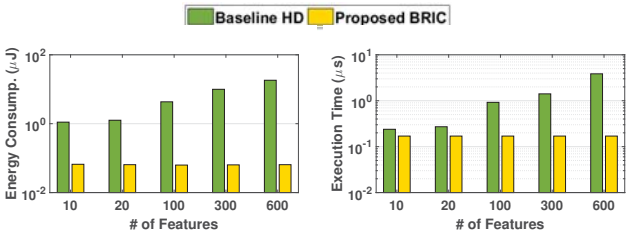

Figure 9: Scalability of the encoding module in BRIC and the baseline HD with the feature size.

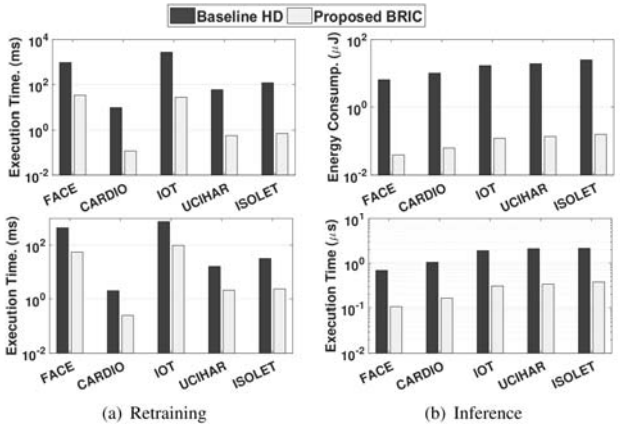

Figure 10: Energy consumption and execution time of BRIC and the baseline HD running (a) a single retraining iteration, and (b) a single query at inference.

since we map each feature vector from n into D dimensional space, where $D >> n$. Let us assume a feature vector with $n = 500$ integer features. For the baseline with integer values, the data size increases by approximately 20×. Even the baseline with a binary encoding ($D = 10,000$) increases the data size by 2.5×, while it provides much lower accuracy. In contrast, the proposed BRIC encodes data points to much lower dimensionality, e.g., $D = 4000$, in order to provide the same accuracy as the baseline. Our evaluation shows that BRIC can ensure 1:1 ratio of high-dimensional data to original data, while providing the same accuracy as baseline HD [9], proving that BRIC is more capable to run on embedded devices with limited memory.

6.3 Hardware Efficiency

We compare the efficiency of BRIC with the state-of-the-art HD computation algorithms on a Kintex-7 FPGA. To have a fair comparison, we consider an optimized implementation of the baseline [9], running on the same architecture as BRIC (explained in Section 5).

Encoding & Training: Due to the predictable memory access pattern and lower BRIC dimensionality, BRIC encoding can process with higher efficiency as compared to the baseline. For instance, to get maximum accuracy, the baseline needs to work with $D = 10,000$ dimensionality while BRIC can provide the same accuracy with $D = 4,000$. Figure 9 shows the scalability of BRIC and the baseline efficiency to the feature size. Our evaluation shows that the execution time of the baseline increases with the number of features, while it takes the same time for BRIC to encode any size feature vector. For applications with 600 features, BRIC provides 282× more energy efficiency and a 22.7× speed up as compared to the baseline.

In training, in order to create class hypervectors, the baseline accumulates integer hypervectors, while BRIC training accumulates binary hypervectors. Figure 8 compares the energy consumption

Table 1: Change in classification accuracy due to online dimension reduction.

Dimension Reduction	20%	40%	60%	70%	80%	90%
FACE	+0.8%	+0.1%	0%	-0.2%	-1.3%	-4.1%
CARDIO	+0.2%	0%	0%	-0.4%	-1.4%	-1.4%
IOT	+0.3%	+0.1%	0%	-14.1%	-16.5%	-21.9%
UCIHAR	+0.6%	-0.1%	-0.8%	-1.2%	-2.1%	-5.4%
ISOLET	0%	-0.4%	-1.9%	-4.6%	-6.9%	-15.2%

and execution time of BRIC and the baseline during initial training. The results are reported when both designs encode and train the model in a pipeline structure. For the baseline, encoding dominates the execution time, thus the training execution hides under the encoding module. However, in BRIC, the encoding can process faster than the training, thus the training is the bottleneck of the execution time (as it is shown in Figure 8). Our evaluation shows that BRIC can provide 64.1× more energy efficiency and a 9.8× speed up as compared to the baseline during training.

Retraining/Inference Efficiency: BRIC stores all encoded hypervectors in order to perform iterative retraining. The existing HD computing algorithms map data points to integer values, where each encoded data is around 20 times larger than the data in the original domain. During retraining, the FPGA needs to sequentially access the encoded values which are pre-stored on off-chip memory. The limited memory bandwidth between the off-chip memory and the FPGA BRAM blocks significantly slows down the baseline computation during retraining. In contrast, BRIC maps the training data to lower dimensions, where each dimension can be represented using a binary value. This enables BRIC to speed up the retraining by loading hypervectors faster than the baseline.

During inference and retraining, HD checks the similarity of each encoded hypervector with all existing class hypervectors. To achieve a high classification accuracy, the existing HD computing algorithms generate an integer model. Therefore, they require the use of an expensive similarity metric such as cosine to find the most similar class. In contrast, BRIC performs the similarity check with hamming distance. Figure 10 shows the energy consumption and execution time of the FPGA accelerating a single retraining iteration and a single query during inference. The results show that BRIC can achieve on average a 61.6× energy efficiency improvement and a 7.9× speed up as compared to the existing HD computation algorithms. Similarly, in inference, the FPGA implementation of BRIC can achieve on average a 43.8× energy efficiency improvement and a 6.1× speed up running a single query (Figure 10b).

6.4 Online Dimension Reduction:

Table 1 shows the impact of the dimension reduction on the BRIC classification accuracy. Our results indicate that dropping 20% of noisy dimensions improves the BRIC accuracy by enabling the model to be retrained based on "significant" dimensions. Further dropped dimensions may result in removing useful information. As listed in Table 1, dropping a larger ratio of dimensions, e.g., 60% of dimensions, results in a 2% quality loss in BRIC accuracy for one of the applications. Online dimension reduction improves BRIC efficiency linearly during both retraining and inference. For example, a 40% dimension reduction results in a 37% energy efficiency improvement and a 29% speed up while providing less than 0.5% quality loss as compared to BRIC with full dimensionality.

7 CONCLUSION

In this paper, we propose BRIC, a novel HD computing framework that significantly improves the computation efficiency. BRIC exploits the predictable memory access of our proposed encoding to design an efficient encoding approach which maps data to a binary hypervector. BRIC enables binary training and retraining on the encoded hypervectors and simplifies the inference similarity metric to Hamming distance. We additionally, implemented a dimension reduction technique that removes unnecessary dimensions to further improve the efficiency of BRIC. We also designed a fully pipelined FPGA implementation to accelerate BRIC. Our evaluations show that BRIC can achieve 64.1× and 9.8× (43.8× and 6.1×) energy efficiency and speed up as compared to the baseline during training (inference) while providing the same classification accuracy.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

[1] J. Gubbi *et al.*, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[2] M. Hassanalieragh *et al.*, "Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges," in *IEEE SCC*, pp. 285–292, IEEE, 2015.
[3] Y. Sun *et al.*, "Internet of things and big data analytics for smart and connected communities," *IEEE Access*, vol. 4, pp. 766–773, 2016.
[4] S. Sicari *et al.*, "Security, privacy and trust in internet of things: The road ahead," *Computer networks*, vol. 76, pp. 146–164, 2015.
[5] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.
[6] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2015.
[7] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *ICRC*, pp. 1–6, IEEE, 2017.
[8] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, ACM, 2016.
[9] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, p. 108, ACM, 2018.
[10] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IoT*, p. 38, ACM, 2018.
[11] P. Kanerva *et al.*, "Random indexing of text samples for latent semantic analysis," in *CogSci*, vol. 1036, Citeseer, 2000.
[12] M. Imani *et al.*, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *DATE*, 2019.
[13] M. Imani *et al.*, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *IEEE BHI*, pp. 271–274, IEEE, 2018.
[14] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *IEEE CLOUD*, pp. 1–6, IEEE, 2019.
[15] A. Rahimi *et al.*, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *ICRC*, pp. 1–8, IEEE, 2016.
[16] T. Wu *et al.*, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *IEEE ISSCC*, IEEE, 2018.
[17] H. Li *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *IEDM*, pp. 16–1, IEEE, 2016.
[18] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," in *IEEE/ACM ICCAD*, pp. 1–7, IEEE, 2018.
[19] M. Imani *et al.*, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *ASPDAC*, pp. 493–498, ACM, 2019.
[20] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, pp. 53–62, ACM, 2019.
[21] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.
[22] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," in *DATE*, 2019.
[23] M. Imani *et al.*, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *IEEE FCCM*, pp. 1–6, IEEE, 2019.
[24] M. Schmuck *et al.*, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *arXiv preprint arXiv:1807.08583*, 2018.
[25] T. I. Cannings *et al.*, "Random-projection ensemble classification," *Journal of the Royal Statistical Society*, vol. 79, no. 4, pp. 959–1035, 2017.
[26] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.
[27] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.
[28] "Uci learning repository." https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities.
[29] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.
[30] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/cardiotocography.
[31] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT.