A Binary Learning Framework for Hyperdimensional Computing

Mohsen Imani[*], John Messerly[*], Fan Wu[‡], Wang Pi[†], and Tajana Rosing[*]
[*]Computer Science and Engineering Department, UC San Diego, La Jolla, CA 92093, USA
[‡]Department of Computer Science and Engineering, UC Riverside, Riverside, CA 92521
[†]School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, P.R.China
{moimani, jmesserl, tajana}@ucsd.edu; piwang@pku.edu.cn

***Abstract*—Brain-inspired Hyperdimensional (HD) computing is a computing paradigm emulating a neuron's activity in high-dimensional space. In practice, HD first encodes all data points to high-dimensional vectors, called hypervectors, and then performs the classification task in an efficient way using a well-defined set of operations. In order to provide acceptable classification accuracy, the current HD computing algorithms need to map data points to hypervectors with non-binary elements. However, working with non-binary vectors significantly increases the HD computation cost and the amount of memory requirement for both training and inference. In this paper, we propose BinHD, a novel learning framework which enables HD computing to be trained and tested using binary hypervectors. BinHD encodes data points to binary hypervectors and provides a framework which enables HD to perform the training task with significantly low resources and memory footprint. In inference, BinHD binarizes the model and simplifies the costly *Cosine* similarity used in existing HD computing algorithms to a hardware-friendly *Hamming* distance metric. In addition, for the first time, BinHD introduces the concept of learning rate in HD computing which gives an extra knob to the HD in order to control the training efficiency and accuracy. We accordingly design a digital hardware to accelerate BinHD computation. Our evaluations on four practical classification applications show that BinHD in training (inference) can achieve 12.4× and 6.3× (13.8× and 9.9×) energy efficiency and speedup as compared to the state-of-the-art HD computing algorithm while providing the similar classification accuracy.**

***Index Terms*—Brain-inspired computing, Energy-efficiency classification, Hyperdimensional computing.**

I. Introduction

Deep Learning has risen to prominence in complex and big data applications. For example, Deep Neural Networks (DNNs) have provided high classification accuracy for complex image classification tasks [1], [2]. However, high computational complexity and memory requirement of DNNs hinder usability to a broad variety of real-life (embedded) applications where the device resources and power budget is limited [3]–[5]. For example, in health care we often require learning algorithms to have a real time control on the patient daily behaviour, speech, and bio-medical sensors [6]–[8]. Sending all data point to the powerful computing environment, e.g., cloud, cannot guarantee scalability and real-time response. It is also often undesirable due to privacy and security concerns [9]–[11]. Thus, we need alternative computing methods that can run a large amount of data at least partly on the less-powerful embedded devices.

Brain-inspired Hyperdimensional (HD) computing has been proposed as a computing method that processes the cognitive tasks in a more light-weight way [12]. HD performs computation on ultra-wide words – that is, with very high-dimensional vectors, or hypervectors. HD works based on the existence of a huge number of nearly orthogonal hypervectors which can be combined using well-defined vector space operations. The mathematics governing the high dimensional space enable HD to be easily applied to different learning problems [13]–[19]. The first step in HD computing is to encode/map data points from the original domain to high-dimensional space. In training, HD combines the encoded hypervectors in order to generate a hypervector representing each class. The classification task at inference performs by checking the similarity of an encoded test hypervector with all trained classes.

In this work, we observe that in order to provide acceptable classification accuracy, the current HD computing algorithms need to encode data points to hypervectors with non-binary elements [18], [20]–[23]. This means that even to perform a single addition between the hypervectors, HD needs to compute thousands (e.g., 10,000) operations. For example, in HD computing, the training performs by the accumulation of several hypervectors. This makes the training operations very expensive. In addition, to perform iterative training, BinHD needs to store all encoded hypervectors which significantly increases the memory footprint. After training, the existing HD computing algorithms generate a model with non-binarized class hypervectors. This model forces the HD inference to use costly *Cosine* metric for similarity check, which involves a large amount of non-binarized additions/multiplications. To reduce the inference cost, prior work tried to binarize the class hypervectors after the training [20], [21], [24]. Although this approach simplifies the inference similarity metric to *Hamming* distance, we observed that it significantly reduces the HD classification accuracy on practical applications, e.g., 11.8% on face recognition application [25].

In this paper, we propose BinHD, a novel framework which enables HD computing to be trained and tested using binary hypervectors. BinHD introduces the concept of learning rate in HD computing by assigning a counter with limited bit-width to each HD model element. BinHD performs training by the accumulation of the binary hypervectors which can process with significantly lower memory footprint and the computation cost. In inference, BinHD removes the necessity of using a non-binary model and costly Cosine similarity by creating a binarized model. This simplifies the HD similarity metric to hardware-friendly Hamming distance. We accordingly design a digital hardware to accelerate BinHD computation in both training and inference. Our evaluation on four practical classification applications shows that BinHD in training (inference) can achieve on average 12.4× and 6.3× (13.8× and 9.9×) energy efficiency and speedup as compared to baseline HD computing algorithm while providing the similar classification accuracy. In addition, we observe that BinHD can provide up to 10.8% higher classification accuracy as compared to the baseline HD computing algorithms [18], [20] using the similar binary hypervectors.

II. Proposed BinHD Training

Here, we proposed BinHD, a framework for binarization of the HD computation during training and inference. Figure 1a shows an overview of BinHD performing the classification task on high-dimensional space. In BinHD, the first step is to map/encode all data points from original to a hypervector, where each element represents using a binary value. Next, the encoded hypervectors are combined in a training module in order to create a single binary hypervector representing each class. In the inference, a test data encodes to high-dimensional space using the same encoding module used for training. Finally, the classification task performs by finding a pre-stored class hypervector which has the highest similarity with the test hypervector. Since BinHD works with a binary model, it enables the inference to use hardware-friendly Hamming distance as a similarity metric. In the following, we explain the details of the HD functionality.

A. BinHD Encoding

BinHD functionality is independent to the encoding module. Here, we consider a general encoding approach which maps a feature vector $F = \{f_1,\ f_2, \ldots,\ f_n\}$, with n features ($f_i \in \mathbb{N}$) to high-dimensional vector $H = \{h_1,\ h_2, \ldots,\ h_D\}$ with D dimensions ($h_i \in \{0,1\}^D$) [12], [18]. Figure 1b shows an overview of the encoding module. This encoding finds the minimum and maximum feature values and quantizes that range into m levels. Then, it assigns a random binary hypervector with D dimension to each quantized level $\{L_1, \ldots, L_m\}$. The level hypervectors are generated such the the neighbor levels have higher similarity, as their absolute values have closer distance [18]. In addition, the encoding module assigns a random binary hypervector to each existing feature index, $\{ID_1, \ldots, ID_n\}$, where $ID \in \{0,1\}^D$. The encoding can happen by linearly combining the feature values over different indices, where a hypervector corresponding to a feature index preserves the position of each feature value in a combined set:

$$H \ = \ ID_1 \oplus \overline{L}_1 \ + \ ID_2 \oplus \ \overline{L}_2 + \ldots \ + \ ID_n \oplus \overline{L}_n.$$

where H is the (non-binary) encoded hypervector, $\oplus$ denotes the XOR operation, and $\overline{L}_i$ is the (binary) hypervector corresponding to the i-th feature of vector F. The binarization of the encoded hypervector can happen by comparing each dimension of H with $n/2$ value. All dimensions with a smaller value than $n/2$ are assigned to 0, while other elements are assigned to 1.

B. BinHD Training

Binary Vector Accumulation: Assume A and B are two binary vectors ($A, B \in \{0,1\}^D$), we define a similar accumulator operation as Sparse Distributed Memory [26], which satisfies the following constraints:

$$\{A' = A[+]B \ \mid \ A, A', B \in \{0,1\}^D \ , \quad \delta(A,B) < \delta(A',B)\}$$

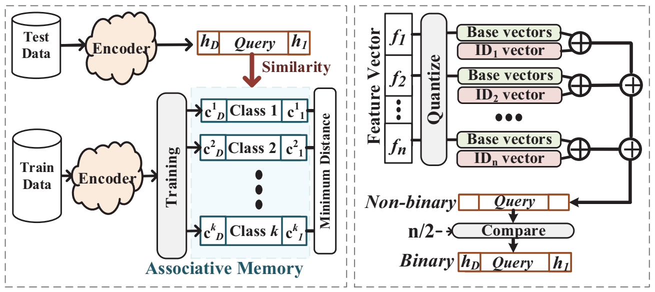


Fig. 1. (a) Overview of HD computing performing the classification task. (b) Functionality of HD encoding module.

Where $[+]$ is a binary addition and $\delta(*)$ is a function that calculates the Hamming distance. It means that for all D dimensional binary vectors A and B, there exists a D dimensional binary vector A' that is the result of accumulating A and B with $[+]$. This satisfies the distance constraint that A' is closer in Hamming space to B than it was to A. The accumulation operation involves representing the accumulator as both a binary vector and as an integer vector of counters. The binary vector keeps the current vector values, while the counter decides to update the binary vector elements during accumulation. Let us assume the binary addition of A and B vectors, where A^C is composed of N-bit counters that saturate at $[-2^{N-1}+1,\ 2^{N-1}]$. While calculating $A[+]B$, the values of A^C counter and the binary vector update as follows:

$$A_i^C = \begin{cases} A_i^C + 1 & B_i > 0 \\ A_i^C - 1 & B_i \leq 0 \end{cases} \longrightarrow A_i = \begin{cases} 1 & A_i^C > 0 \\ 0 & A_i^C \leq 0 \end{cases}$$

Counter Update **Binary Vector Update**

Figure 2 shows an example of the accumulation operation. The accumulation of A and B binary vectors performs in two steps. First, a counter update, where A^C updates depending on B vector elements. The B elements with "1" value increment the A^C counter, while "0" elements decrement A^C. For the example shown in Figure 2b, we use $N = 5$, thus counter values saturates between -15 and +16 range. The second step is updating the binary vector A, depending on changes on A^C. As Figure 2c shows, the value of the accumulator vector, A, flips on all dimensions that the counter values changed from positive to zero/negative or vice versa. We explore the impact of counter size on Section II-D.

Initial Model Generation: We accumulate all binary encoded hypervectors in training dataset to create k binary prototype vectors $\{C_1, \cdot, C_k\}$, where k is number of classes and $C_i \in \{0,1\}^D$. These prototype vectors represent the average (centroid) of that particular class, with respect to Hamming space. We can view each class hypervector as a linear combination of the encoded hypervectors in that class. For example, i^{th} class hypervector can be computed as: $C_i = \sum_{\forall j \in class_i} H_j$. As we explained, the accumulation of the binary hypervectors happens by assigning a counter vector to each existing class (e.g., C_i^{count} for class i^{th}). The counter keeps track of the number of 0s and 1s in each class dimension and assigns C^i dimension to 1 if the number of 1s exceeds 0s. After initialization, the model is ready for the classification. The inference is performed by taking the Hamming distance between the query data, and all k class hypervectors. We label the query as a class with the highest Hamming distance.

HD model initialization comes with two advantages: first, by making a strong assumption about how each class is an average of the training data, our classifier is no longer a black box model that requires iterative convergence on the training data, but a prototype model that can be generated in a single pass. This initial model often provides acceptable accuracy, and gradient descent becomes an optional optimization, rather than a necessary obstacle. Second, for gradient descent, these initial model vectors give vital information about which dimensions are the more significant than others, which is information that would have otherwise been lost through binarization. We further discuss it on Section II-C.

C. *BinHD Model Adjustment*

We can significantly reduce the error rate of the initial HD model by employing gradient descent. We propose an online stochastic approach to descent. Figure 3 shows the overview of BinHD functionality during model adjustment. BinHD first encodes all training data point into high-dimensional binary vectors (Figure 3a). For each incoming sample of training data, our approach attempts to classify it by measuring its Hamming distance with the trained model (Figure 3b). If the model with the highest Hamming distance matches the correct label, BinHD ignores updating the model. However, if a train data point, H, incorrectly matches with HD model, we add the query to a correct class (C_i), while subtracting it from an incorrect one (C_j).

$$C_i = C_i[+]H \quad \& \quad C_j = C_j[-]H$$

Since the query and class hypervectors are both binary, the addition ($[+]$) and subtraction ($[-]$) happen by updating the corresponding counter array shown in Figure 3c. These counters update using the accumulation definition introduced in Section II-B. For example, $C_i[+]H$, increments/decrements the class counter values on all dimensions that H has 1/0 values. After updating the counters, BinHD flips the class elements on all dimensions that their counter values are changed from positive to zero/negative values or vice versa (Figure 3d). The model adjustment continues until for a pre-defined number of iterations (maximum 30 iterations), unless if the accuracy converges earlier. The convergence condition is having less than $\varepsilon = 0.1\%$ change in accuracy in three consecutive iterations.

BinHD model adjustment has two main advantages as compared to the existing HD computing algorithms [18], [21]. First, unlike existing approaches which need to store non-binary encoded hypervectors, BinHD stores binary encoded hypervectors with significantly lower memory size. Second, BinHD exploits a binary model with Hamming distance similarity during retraining, while the computation of prior work depends on the non-binary model with costly Cosine similarity [21]. These facts make BinHD an efficient light-weight classification approach for embedded devices with limited resources.

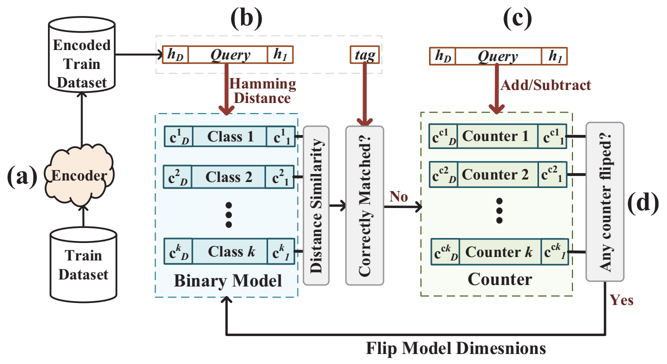


Fig. 3. BinHD model adjustment in binarized domain.

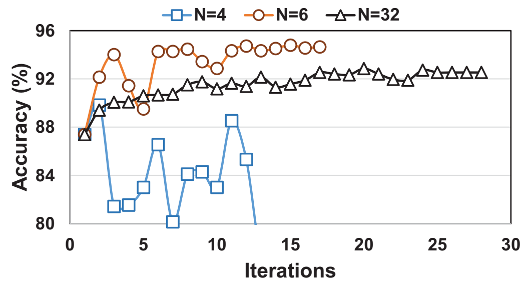


Fig. 4. Impact of the counter size on the BinHD classification accuracy.

D. *Controlling the Learning Rate*

The learning rate is a crucial parameter in stochastic gradient descent that controls the step size. In most applications, the learning rate is a floating point value between 0 and 1 that is multiplied by training samples before they are added or subtracted from the vector of weights. The intuition is that this regulates how much individual training samples are allowed to move the classifier's hyperplane. Learning rates that are too small allow for the descent to get trapped in local optima while learning rates that are too large cause the descent to diverge.

In the Hamming Distance classifier's, such as BinHD, the learning rate is more difficult to define. The binary weights of the model vectors only flip when the counters cross the zero thresholds. This means that there is no guarantee that adding a single binary vector to the counters will take any immediate effect on the model. We choose to define our learning rate as the average number of bits flipped per accumulation operation. We modify this rate by saturating the counters with a reasonable ceiling and increasing the number of counter increments per accumulation. In fact, the size of a counter determines an application learning rate.

Figure 4 shows the impact of counter size on the BinHD classification accuracy during the model adjustment iterations. The results are reported for activity recognition dataset [27] using three different counter sizes. BinHD using 32-bit counters results on average 2-5 bits flipped per accumulation. It means that the training samples are only allowed to modify the vector about 0.05% per accumulation. This is unsurprisingly low since using counters that can accommodate values as high as 6,213 or as low as -6,213 (the number of training samples in the set), some dimensions will take thousands of

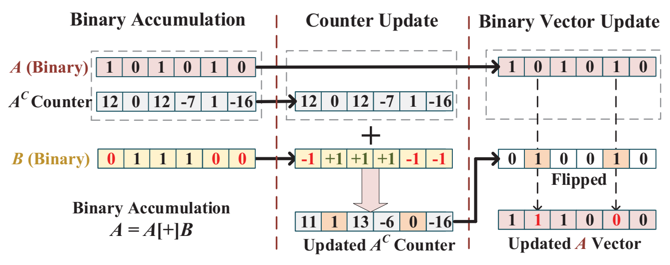


Fig. 2. Example of the binary accumulation using counter thresholding.

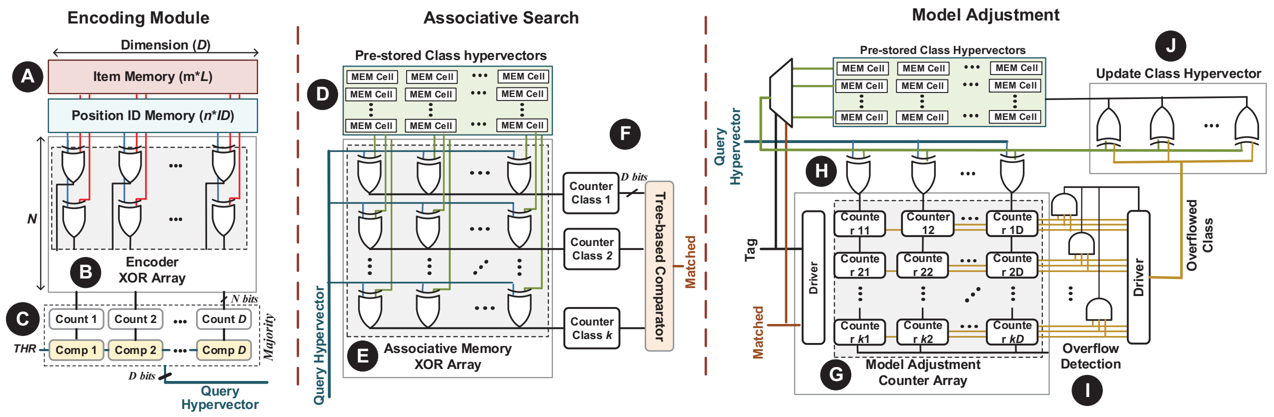


Fig. 5. The hardware implementation of BinHD including encoding, associative search, and model adjustment modules.

accumulations to flip. Convergence on the training data will take hundreds of epochs with such a low learning rate. BinHD can increase the learning rate by using a smaller counter size. For example, using counter size equal to 6-bits ($N = 6$) increases the learning rate and results in higher classification accuracy. In fact, 6-bits counter size is equivalent of using a learning rate of 5%, which results in a faster training. From the other side, using very small counter size, i.e., 4-bit, is equivalent to use very large learning rate which increases accuracy fluctuation and increases the chance of divergence.

III. Hardware Implementation

The main computation of HD can perform using three main blocks: encoding, associative search and counter modules. Figure 5 shows the details implementation of these three blocks.

A. Encoding

Figure 5 shows the implementation of encoding module. Our approach stores all position (ID) and level (L) hypervectors in position and item memory blocks respectively (A). After access to the feature values in the original domain, BinHD compares each feature value with the quantized feature values. Each feature is assigned to a quantized level which it has a minimum distance with. BinHD reads a level hypervector from the memory and XORs it with the position hypervector corresponding to that feature (B). This process can perform in parallel for all features. The result of XOR operations are accumulated using D counter blocks and compared with a threshold value. In BinHD, a threshold value is the half of the number of features ($THR = n/2$). This results in generating an encoded hypervector with D binary elements (C).

B. Training

In the existing HD computing algorithms, the training happens by accumulating the non-binarized hypervectors, resulting in large memory requirement and expensive computational cost [18]. BinHD implements training by accumulating all encoded hypervectors corresponding to a class. Since the encoded hypervectors are binary, this addition happens using the binary accumulation approach introduced in section II-B. BinHD uses a counter array for each corresponding class which keeps track of the number of 0 and 1 bits in each dimension (Figure 5G). After updating the counter values for the entire training data, BinHD creates an initial training model by assigning any dimensions with positive value to 1, while counters with zero or negative values are assigned to 0.

C. Associative Search

In inference and model adjustment, the associative search is the main cost of HD computing where we compare the similarity of an encoded hypervector with a binarized HD model. Unlike prior HD computing algorithms that use costly Cosine as the similarity metric, BinHD performs similarity check using Hamming distance. BinHD pre-stores the trained class hypervectors in a memory block (D). The similarity check of a query and class hypervectors performs using an array of XOR gates (E). Each XOR row computes the Hamming distance similarity of a query and class hypervector. A counter block has been located at the right side of the array is responsible to count the number of mismatches in each class. Finally, a tree-based comparator block identifies a class with the minimum Hamming distance (F).

D. Model Adjustment

The model adjustment can be implemented using the same XOR array used for similarity check and a counter array used for initial training (G). As we explained in section II-C, BinHD uses a single N-bits counter to keep track of changes in each dimension of a class hypervector. For example, for an application with k classes and D dimensions, we require $k \times D$ counters, where each counter corresponds to a single dimension of a class hypervector. Model adjustment block uses the same XOR array to check the similarity of a training data point with the HD model. Depending on the correctness of match, BinHD implements the model adjustment using the following steps:

- If an encoded hypervector matches with a correct class, BinHD continues the search for the next data point without updating the counter array.
- If an encoded hypervector match with an incorrect class hypervector, BinHD XORs the query/encoded hypervector with that class (H). Model adjustment accesses to the class

TABLE I
Datasets (n: feature size, k: number of classes).

	n	K	Train Size	Test Size	Description
ISOLET	617	26	6,238	1,559	Speech recognition [28]
UCIHAR	561	12	6,213	1,554	Activity recognition(Mobile) [27]
FACE	608	2	522,441	2,494	Face recognition [25]
CARDIO	21	3	1,913	213	Cardiotocograms classification [29]

TABLE II
COMPARING THE CLASSIFICATION ACCURACY, TRAINING MEMORY FOOTPRINT, AND MODEL SIZE OF BINHD AND THE BASELINE HD COMPUTING IN DIFFERENT CONFIGURATIONS.

Encoding/Model	Classification Accuracy				Training Memory Footprint (MB)				Model Size (KB)			
	ISOLET	UCIHAR	FACE	CARDIO	ISOLET	UCIHAR	FACE	CARDIO	ISOLET	UCIHAR	FACE	CARDIO
Float/Float HD [18]	93.5%	95.8%	95.3%	99.0%	251	249	898	77	1,015.6	468.7	78.1	117.2
Float/Binary HD [20]	88.1%	91.3%	91.9%	93.8%	251	249	898	77	31.7	14.6	2.4	3.7
Binary/Binary HD [20]	85.6%	87.3%	83.5%	90.2%	10	13	34	3	31.7	14.6	2.4	3.7
Proposed BinHD	91.5%	95.7%	94.3%	99.5%	10	13	34	3	31.7	14.6	2.4	3.7

hypervector using the same memory block which stored the trained class hypervectors. Every element of an `XOR` vector with 1 (0) bit increments (decrements) the counter value of the corresponding class. BinHD also updates the counter values on a class that the data point belongs to by using a *Tag* control signal. BinHD `XOR`s the encoded hypervector with the correct class hypervector. Depending on the result of `XOR`, we increment (decrement) the counter values on the dimensions that `XOR` results have 0 (1) bit.

During the above update steps, if a sign of a counter changes, our approach fillips the corresponding dimensions of class hypervector. This update happens in two steps: detecting any changes in the counters corresponding to a class by `AND`ing their signs signals (I). Second, `XOR`ing the sign signal with the corresponding class and write the results back to the same memory location (J).

IV. EVALUATION

A. Experimental Setup

We verified the functionality of BinHD using both software and hardware implementations. In software, we implement HD training and inference on Intel Core i7 7600 CPU using an optimized C++ implementation. For a hardware implementation, we use a standard digital ASIC flow to design dedicated hardware. We describe HD functionality using RTL System-Verilog. For the synthesis, we use *Synopsys Design Compiler* with the TSMC 45 nm technology library, the general purpose process with high V_{TH} cells. We extract its switching activity during post-synthesis simulations in *ModelSim* by applying the test sentences. We compare the BinHD efficiency and accuracy with the state-of-the-art HD computing algorithm proposed in [18], [21]. Table I summarizes the evaluated datasets. The tested benchmarks range from relatively small datasets collected in a small IoT network, e.g., UCIHAR, to a large dataset which includes hundreds of thousands of images of facial and non-facial data.

B. BinHD vs Existing Algorithms

Table II compares the classification accuracy of BinHD with the baseline HD computing algorithm [18] in three different configurations. First, the baseline HD using non-binary encoding and non-binary model (Float/Float) which provides the highest classification accuracy. Second, HD computing with non-binary encoding and binary model (Float/Binary). Third, HD computing with binary encoding and binary model (Binary/Binary) which has the maximum efficiency. Our evaluation shows that HD computing in Float/Float configuration provides on average 6.7% and 9.3% higher classification accuracy as compared to HD in Float/Binary and Binary/Binary configurations respectively. In fact, naively binarization of the encoded hypervector or HD model results in a significant drop in the classification accuracy. In contrast, our proposed BinHD framework binarizes both encoded and class hypervectors with minimal impact on the classification accuracy. Our evaluation shows that BinHD can provide the similar accuracy to the baseline HD in Float/Float configuration (less than 0.6%).

Table II also compares proposed BinHD and the baseline HD computing algorithms in terms of training memory footprint and model size. The baseline HD computing algorithm encodes data points to non-binary hypervectors, thus they require large memory footprint during training. BinHD enables HD computing to work with binarized encoded hypervectors. Our evaluation shows that BinHD on average requires 24.6× lower memory footprint as compared to the baseline HD using non-binary encoded hypervectors. In terms of model size, BinHD provides the same memory size as HD with the binarized mode, which is 32× smaller than the baseline HD with the non-binary model. In summary, BinHD can provide the memory/computing efficiency of the fully binary HD (Binary/Binary) as well as the classification accuracy of the baseline HD with non-binary encoding and model (Float/Float).

C. BinHD & Counter Size

Table III lists the classification accuracy of BinHD when the counter width increases from 6-bits to 32-bits. Choosing the width of the counters, like choosing a learning rate, depends on the dataset size. Choosing a counter that is small does not provide enough memory for retraining to converge on large datasets. In another word, accumulating enough changes cause forgetfulness during the descent. Our results in Table III shows that using counters smaller than 10-bits results in a divergence of the FACE recognition application. Similarity, the accuracy of the other applications diverge when the counter is smaller than 4-bits. Choosing a counter width that is too large will not give the significant dimensions a high enough probability to be flipped. Therefore, the gradient descent will choose to fit the data based on insignificant features, leading to overfitting. Our evaluation shows that the best counter size is predictable depending on the dataset size. Depending on the number of train data, we should select a counter size which provides a learning rate of 5%. For example, for a large dataset such as FACE, BinHD requires to use 10-bits counter size, while for smaller datasets such as ISOLET and UCIHAR using 6-bit counters provides 5% learning rate.

D. Training Efficiency

Here, we compare the efficiency of BinHD and the baseline HD computing algorithm with Float/Float configuration which provides the similar accuracy as BinHD. The baseline HD encodes data point to the non-binary domain and then adds the encoded hypervectors in order to create each class hypervector. BinHD simplifies the training operation by performing accumulation of the binary encoded hypervectors. Figure 6 compares the energy consumption and execution time of BinHD and baseline HD running on digital hardware. Our

TABLE III
IMPACT OF THE COUNTER SIZE ON BINHD CLASSIFICATION ACCURACY.

Counter Size	5-bits	6-bits	8-bits	10-bits	16-bits	32-bits
ISOLET	91.7	91.7	91.5	91.2	90.5	89.2
UCIHAR	95.8	95.7	94.3	93.1	93.7	92.5
FACE	NA	NA	NA	94.2	93.1	92.4
CARDIO	99.5	99.5	99.1	97.6	95.8	97.6

evaluation shows that BinHD can provide 6.3× faster and 12.4× higher energy efficiency as compared to the baseline HD computing algorithm [18], while providing the similar classification accuracy.

E. Testing and Model Adjustment Efficiency

Figure 7a compares BinHD and the baseline HD computing algorithm during inference. Since BinHD uses a binary model, it can exploit a hardware-friendly Hamming distance as the similarity metric, while the baseline HD using non-binary model uses costly Cosine for the similarity check. Unlike Hamming distance, calculating Cosine similarity is so costly as it involves a large number of non-binary multiplications. Our evaluation shows that BinHD can achieve 13.8× higher energy efficiency and 9.9× speedup as compared to the baseline HD using non-binary model.

Figure 7b shows the efficiency of BinHD and the baseline HD computing during a single iteration of model adjustment. The retraining consists of an associative search and model update. Similar to the inference, BinHD provides significantly higher efficiency than the baseline HD computing. This is because the retraining in HD performs by applying similarity check to the binary model, while the baseline HD needs to use costly Cosine similarity on the non-binarized model. The model update in BinHD can perform using bitwise `XOR` operation, while the baseline HD requires to perform non-binary addition on two class hypervector. Our evaluation shows that BinHD can achieve on average 13.6× speedup and 7.8× higher energy efficiency as compared to the baseline HD computing algorithms.

V. CONCLUSION

In this paper, we proposed a novel framework for binarization of the Hyperdimensional computing algorithm during training and inference. BinHD encodes data points into binary hypervectors and performs training using the binary accumulation. In the inference, BinHD creates a binary model which enables the computation happens using light-weight Hamming distance similarity check. Our evaluation shows that BinHD in training (inference) can achieve on average 12.4× and 6.3× (13.8× and 9.9×) energy efficiency and speedup as compared to baseline HD computing algorithm while providing the similar 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.

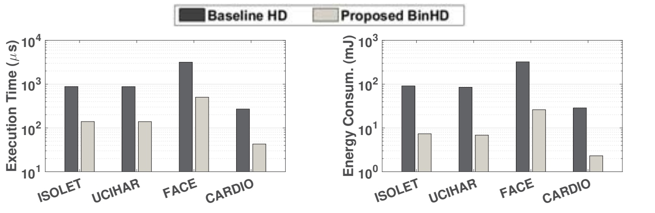
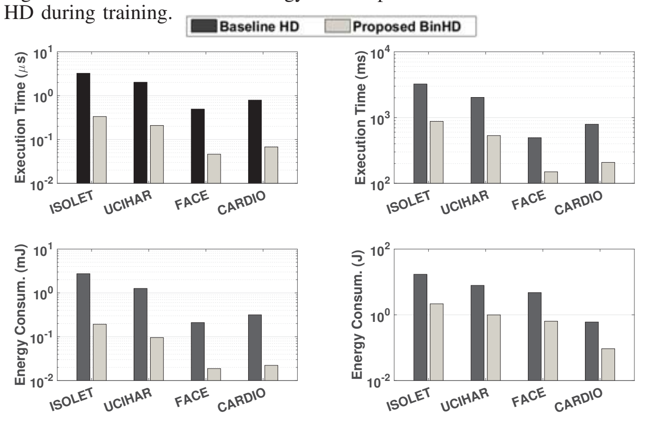


Fig. 6. Execution time and energy consumption of BinHD and the baseline HD during training.



(a) Testing (b) Model Adjustment

Fig. 7. Execution time and energy consumption of BinHD and the baseline HD running (a) a single query in the inference (b) a single iteration of the model adjustment.

REFERENCES

[1] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
[2] M. Imani *et al.*, "Bayesian control of large MDPs with unknown dynamics in data-poor environments," in *NIPS*, 2018.
[3] M. Denil *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, pp. 2148–2156, 2013.
[4] D. Bouris *et al.*, "Fast and efficient fpga-based feature detection employing the surf algorithm.," in *FCCM*, vol. 10, pp. 3–10, 2010.
[5] M. Imani *et al.*, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.
[6] G. Surrel *et al.*, "Online obstructive sleep apnea detection on medical wearable sensors," *IEEE Transactions on Biomedical Circuits and Systems*, no. 99, pp. 1–12, 2018.
[7] M. Chen *et al.*, "Disease prediction by machine learning over big data from healthcare communities," *IEEE Access*, vol. 5, pp. 8869–8879, 2017.
[8] D. Sopic *et al.*, "Real-time classification technique for early detection and prevention of myocardial infarction on wearable devices," in *BioCAS*, pp. 1–4, IEEE, 2017.
[9] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 70–73, 2014.
[10] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.
[11] G. Hatzivasilis *et al.*, "Secroute: End-to-end secure communications for wireless ad-hoc networks," in *ISCC*, pp. 558–563, IEEE, 2017.
[12] 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.
[13] M. Imani *et al.*, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *ASP-DAC*, IEEE, 2019.
[14] 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.
[15] M. Imani *et al.*, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design & Test*, vol. 34, no. 6, pp. 94–101, 2017.
[16] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.
[17] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, ACM, 2019.
[18] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, p. 108, ACM, 2018.
[19] M. Imani *et al.*, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *BHI*, pp. 271–274, IEEE, 2018.
[20] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, ACM, 2016.
[21] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *ICRC*, pp. 1–6, IEEE, 2017.
[22] A. Moin *et al.*, "An emg gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," in *ISCAS*, pp. 1–5, IEEE, 2018.
[23] S. Gupta *et al.*, "Felix: fast and energy-efficient logic in memory," in *ICCAD*, p. 55, ACM, 2018.
[24] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IoT*, p. 38, ACM, 2018.
[25] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *ICCAD*, pp. 25–32, IEEE, 2017.
[26] Y. Wu *et al.*, "The kanerva machine: A generative distributed memory," *arXiv preprint arXiv:1804.01756*, 2018.
[27] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *IWAAL*, pp. 216–223, Springer, 2012.
[28] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.
[29] D. Ayres-de Campos *et al.*, "Sisporto 2.0: a program for automated analysis of cardiotocograms," *Journal of Maternal-Fetal Medicine*, vol. 9, no. 5, pp. 311–318, 2000.