ManiHD: Efficient Hyper-Dimensional Learning Using Manifold Trainable Encoder

Zhuowen Zou*, Yeseong Kim‡, M. Hassan Najafi$^{\Psi}$, Mohsen Imani†⋆
*University of California San Diego, ‡DGIST, $^{\Psi}$University of Louisiana, †University of California Irvine
⋆Email: m.imani@uci.edu

***Abstract*—Hyper-Dimensional (HD) computing emulates the human short memory functionality by computing with hyper-vectors as an alternative to computing with numbers. The main goal of HD computing is to map data points into sparse high-dimensional space where the learning task can perform in a linear and hardware-friendly way. The existing HD computing algorithms are using static and non-trainable encoder; thus, they require very high-dimensionality to provide acceptable accuracy. However, this high dimensionality results in high computational cost, especially over the realistic learning problems. In this paper, we proposed ManiHD that supports adaptive and trainable encoder for efficient learning in high-dimensional space. ManiHD explicitly considers non-linear interactions between the features during the encoding. This enables ManiHD to provide maximum learning accuracy using much lower dimensionality. ManiHD not only enhances the learning accuracy but also significantly improves the learning efficiency during both training and inference phases. ManiHD also enables online learning by sampling data points and capturing the essential features in an unsupervised manner. We also propose a quantization method that trades accuracy and efficiency for optimal configuration. Our evaluation of a wide range of classification tasks shows that ManiHD provides 4.8% higher accuracy than the state-of-the-art HD algorithms. In addition, ManiHD provides, on average, 12.3× (3.2×) faster and 19.3× (6.3×) more energy-efficient training (inference) as compared to the state-of-the-art learning algorithms.**

I. Introduction

In 2025 more than 175 Zettabytes of data will be generated worldwide, much of it by machines, but less than 1% will be analyzed. We are drowning in data. Today's systems rely on sending all the data to the cloud, and then using complex algorithms, such as Deep Neural Networks, which require billions of parameters and many hours to train [1], [2]. This computational cost is beyond the capability of today's embedded devices, which often have limited resources and battery [3]. Therefore, we need alternative learning methods to train on the less-powerful devices while providing good enough classification accuracy.

The human brain can do much of this learning effortlessly. In particular, Hyperdimensional (HD) computing has been proposed as a light-weight brain-inspired learning methodology [4], [5], [6]. HD computing is motivated by the observation that the human brain operates on high dimensional representations of data [4]. This high-dimensional space is referred to as a hyperspace, while points in the space are known as hypervectors. Because of their high-dimensionality, any randomly chosen pair of hypervectors will be nearly orthogonal (e.g., uncorrelated). HD computing provides several features that make it suitable for efficient learning in IoT systems. First, HD models are computationally efficient (highly parallel at heart) to train and amenable to hardware level optimization [7], [8], [9], [10]. Second, HD models offer an intuitive and human-interpretable model and offer a complete computational paradigm that can be applied to cognitive and learning problems [11], [12]. Finally, it provides strong robustness to noise – a key strength for IoT systems. These features make HD a promising solution for today's embedded devices with limited storage, battery, and resources [13], [14], [15].

HD computing performs the learning task after encoding all data points to high-dimensional space. The required dimensionality of HD computing increases based on the complexity of the classification takes. All existing HD computing methods use a static and non-trainable encoder to map data into high dimensional space [7], [6], [5], [16]. In other words, the encoding module does not learn or differentiates between the features in the input data. Therefore, HD computing requires very high dimensionality in order to solve realistic learning problems. However, higher dimensionality results in low computational efficiency.

In this paper, we proposed ManiHD, a trainable encoder for efficient adaptive learning in high-dimensional space. ManiHD enhances the randomized HD encoder with manifold learning in order to eliminate the extreme dimensionality of hypervectors. The main contributions of the paper are as follows:

- To the best of our knowledge, ManiHD is the first HD computing approach that provides adaptive and trainable encoding. Instead of using a static encoder, ManiHD explicitly considers non-linear interactions between the features during the encoding. This enables ManiHD to provide maximum learning accuracy using much lower dimensionality.
- ManiHD enables online learning by frequently sampling data and capturing the important features in an unsupervised manner. ManiHD enables to adapt itself to changes in the input data or environment during the prediction phase.
- To reduce the ManiHD computation cost, we also introduce the idea of quantizing the encoding module. In addition, to compensate for the quality loss caused by quantization, while providing an optimal system efficiency.

ManiHD dimension reduction results in significant improvement in the efficiency of training and inference phases. We evaluation ManiHD on a wide range of classification problems. Our evaluation on a wide range of classification tasks shows that ManiHD provides 4.8% higher accuracy than the state-of-the-art HD algorithms. In addition, ManiHD provides, on average, 12.3× (3.2×) faster and 19.3× (6.3×) more energy-efficient training (inference) as compared to the state-of-the-art learning algorithms.

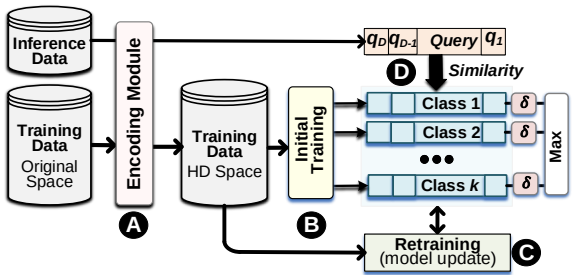

Fig. 1. Overview of HD computing for classification.

II. ManiHD Classification

Figure 1 shows the overview of the HD classification. In the first step, HD computing performs the learning task after mapping all training data into the high-dimensional space [7], [5], [17]. To find the universal property for each class in the training dataset, we combine hypervectors belonging to each class, i.e., adding the hypervectors to create a single hypervector for each class. Once combining all hypervectors, we treat per-class accumulated hypervectors, called *class hypervectors*, as the learned model. Next, a similarity search procedure performs the inference task. For a given *query* hypervector encoded for a tested data point, it selects the class that has the most similar class hypervector. In the following, we explain the details of the learning process and related work.

A. Classification in HD Space

Ⓐ Encoding: The encoded data should satisfy the commonsense principle: data points which are different from each other in the original space should be also different in the high-dimensional space. There are multiple encoding methods proposed in literature [18], [19], [7]. Although these methods have shown excellent classification accuracy for their application-specific problems, to the best of our knowledge, the existing encoding methods linearly combine the hypervectors corresponding to each feature, resulting in sub-optimal classification quality for general classification problems. To obtain the most informative hypervectors, the HD encoding should consider the non-linear interactions between the feature values with different weights.

Ⓑ Training: In the training step, HD combines all the encoded hypervectors of each class using the element-wise addition. For example, in an activity recognition application, the training procedure adds all hypervectors which have the "walking" and "sitting" tags into two different hypervectors. Where $\mathbf{H}^i_j = \langle h_D, \cdots, h_1 \rangle$ is encoded for the j^{th} sample in i^{th} class, each class hypervector is trained as follows: $\mathbf{C}^i = \sum_j \mathbf{H}^i_j = \langle c^i_D, \cdots, c^i_1 \rangle$

If the encoding method projects the original data non-linearly to the high dimensional space, the linearly combined model can perform well even on non-linearly separable data.

Ⓒ Retraining: Once the initial training is done, we train the class hypervector model again to improve the classification accuracy. In this *retraining* step, we calculate the similarity between each encoded hypervector and trained model to check whether the data sample is correctly classified or not. If the encoded hypervector, $\mathbf{H}$, is correctly classified by the current model, we will make no changes to the model. Otherwise, we update the model by respectively adding and subtracting it from the correct and incorrect classes as follows:

$$\overline{\mathbf{C}}^{correct} = \mathbf{C}^{correct} + \alpha\mathbf{H} \quad and \quad \overline{\mathbf{C}}^{wrong} = \mathbf{C}^{wrong} - \alpha\mathbf{H}$$

The retrained model provides a better fit to the training data and gets higher accuracy. We repeat the same procedure for multiple iterations. In our observation, repeating 20 iterations yields sufficient convergence for all the tested datasets.

Ⓓ Inference: The main computation of the inference is the encoding and associative search. We perform the same encoding procedure to convert a test data point into a hypervector, called *query hypervector*, $\mathbf{Q} \in \{0,1\}^D$. Then, it computes the similarity of the query hypervector with all k class hypervectors, $\{\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_k\}$. We measure the similarity between a query and a i^{th} class hypervector using: $\delta\langle \mathbf{Q}, \mathbf{C}_i \rangle$, where δ denotes the similarity metric. After computing all similarities, each query is assigned to a class with the highest similarity.

B. Challenges

The existing HD algorithms are using a static encoder to map data into high-dimensional space. We observe that this is the main reason that HD requires high dimensionality to provide acceptable accuracy. However, the HD computation cost increases with hypervector dimensionality. In this paper, we propose the idea of adaptively and trainable encoder which enables HD computing to provide maximum accuracy in much lower dimensionality.

III. ManiHD Adaptive Encoding

This section proposed ManiHD, an HD-based classification algorithm supporting an adaptive encoder for efficient learning in high-dimensional space. ManiHD encoder is built based on two goals: (i) an encoder that can provide high classification accuracy in smaller dimensions, (i) a dynamic encoding module that can adaptively change depending on data and environment. ManiHD introduces a novel HD encoder that considers the relationship between different features before mapping data into high-dimensional space. ManiHD samples a small portion of training data to determine the relationship between different features in an unsupervised way. Then, it modifies the HD encoder to consider such a relationship before encoding the data point. ManiHD provides several interesting features: (i) eliminates the necessity of using a very large naive projection matrix as an encoding module, (ii) provides high classification accuracy even using lower dimensionality.

A. Non-Linear Encoding

In this context, we propose a novel encoding method which exploits the kernel trick to map data points into the high-dimensional space. The proposed encoding method is inspired by the Radial Basis Function (RBF) kernel trick method [20], [21]. Figure 2A shows our encoding procedure. Let us consider an encoding function that maps a feature vector $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$, with n features ($f_i \in \mathbb{N}$) to a hypervector $\mathbf{H} = \{h_1, h_2, \ldots, h_D\}$ with D dimensions ($h_i \in \{-1, 1\}$). We generate each dimension of the encoded data by calculating a dot product of the feature vector with a randomly generated vector as $h_i = cos(\mathbf{B}_i \cdot \mathbf{F})$, where B_i is the randomly generated vector with a Gaussian distribution (mean $\mu = 0$ and standard

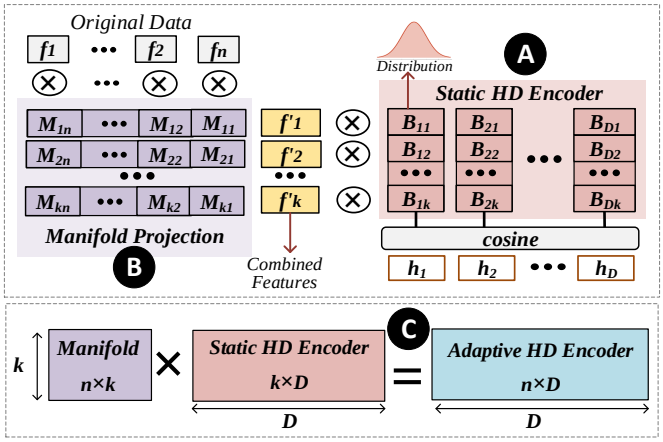

Fig. 2. ManiHD encoder consisting of manifold and non-linear HD encoder.

deviation $\sigma = 1$) with the same dimensionality to that of the feature vector. The random vectors $\{\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_D\}$ can be generated once offline and then can be used for the rest of the classification task. After this step, each element h_i of a hypervector $\mathbf{H}^n$ has a non-binary value. In HD computing, binary (bipolar) hypervectors are often used for computation efficiency. We thus obtain the final encoded hypervector by binarizing it with a sign function ($\mathbf{H} = sign(\mathbf{H}^n)$) where the sign function assigns all positive hypervector dimensions to '1' and zero/negative dimensions to '-1'. The encoded hypervector stores the information of each original data point with D bits.

B. Manifold Learning

Here, we exploit manifold learning to capture non-linear structure in data before naively encoding it into high-dimensional space. Manifold learning is an approach to non-linear dimensionality reduction, assuming in many data sets, the number of input features is artificially high. Manifold algorithms are unsupervised [22], [23]; thus, they can learn the data structure without using any labeled data or predetermined classifications. Our goal is to combine the capability of manifold learning with ManiHD static encoder to design an adaptive and trainable encoder for high-dimensional classification.

Although there are multiple manifold learning methods, ManiHD requires an approach supporting online and low-cost transformation for the data. Here, we focus on `isomap` (i.e., Isometric mapping) [24], [22]. The goal of `isomap` is to find a lower-dimensional of embedding that keeps the geometric distances between all data points. `isomap` uses the nearest search operation and shortest-path graph search to find a proper projection from original data to a lower dimension. This projection considers the relation between the features and generates a new shorter vector between a representation for learning purposes.

In `isomap`, the cost of manifold learning increases quadratically with the number of data points ($O[N^2]$). Therefore, running manifold learning over the entire training data results in a significantly slow training process. One of the main goals of HD computing is to enable fast on-the-fly training on low-end embedded devices. Therefore, the high computational cost of manifold learning does not allow us to run it during the training phase. To address this issue, as we show in Figure 3,

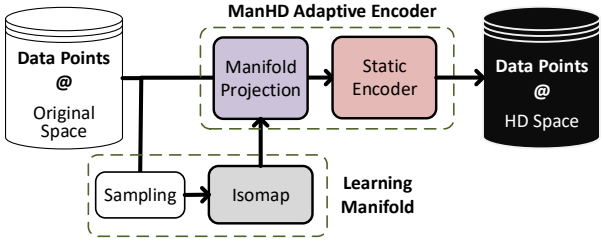

Fig. 3. ManiHD framework for online trainable encoder.

ManiHD randomly samples a small portion of data points (e.g., about 1%) to learn the manifold. The output of the algorithm is a projection matrix that maps input data to lower dimensions ("Manifold Projection" shown in Figure 2B).

For all data points (training/test data), ManiHD first passes each original data ($\{f_1, f_2, \cdots, f_n\}$) through a manifold projection matrix. The result will be a vector with a smaller dimensionality, $\{f'_1, f'_2, \cdots, f'_k\}$, where $k < n$. Next, ManiHD runs the HD static encoder on the manifold output vector in order to map it into high-dimensional space. The encoded data will be binarized and then used for the rest of the training and inference tasks. As Figure 2C shows, the manifold projection adds an extra cost to ManiHD static encoder. In other words, ManiHD adaptive encoder consists of two vector-matrix multiplication: first, the manifold projection matrix and second, HD static projection. This extra cost can affect the efficiency of both training and inference phases, as every test data also needs to pass through these two matrices. To address this overhead issue, ManiHD combines manifold and HD projection matrix into a single matrix. As Figure 3C shows, we multiply these matrices once after training the manifold and use a new adaptive HF encoder for encoding the rest of the data points. Note that this matrix's size is the same as the original encoding matrix ($n \times D$); thus, computationally, it does not add any extra cost to the HD static encoding.

IV. ManiHD Online Learning

In real-world learning problems, data are changing over time as the environment is dynamic. Having a static encoder which has been pre-trained once does not give flexibility to ManiHD. In addition, ManiHD requires fast and efficient encoding and learning in order to enable real-time learning. In this section, we explain how ManiHD supports both these features.

A. Real-time Encoding Update

In order to design a flexible and dynamic encoder, ManiHD processes manifold frequently learning over input data. Since the manifold is unsupervised, it can be processed on the unlabeled data during the inference. After a pre-defined time, ManiHD samples from new data points and run manifold learning on a small portion of data. Next, it updates the HD encoder based on a new projection matrix. This enables us to adaptively improve our encoder's quality, depending on the input data changes. Since manifold learning is computationally expensive, we can run the encoding offline and only once after a while to make sure we do not increase the computational training cost. In this scenario, ManiHD can use the old projection matrix to perform a learning task while updating the projection matrix based on new data. In Section V-E,

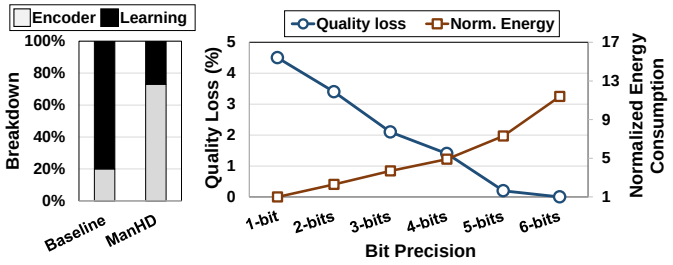

Fig. 4. Quality vs efficiency trade-off.

we explore the impact of the frequent manifold update on ManiHD accuracy and efficiency.

B. ManiHD Encoder Qunatization

In order to enable online learning, ManiHD needs to provide real-time data encoding and learning. However, ManiHD encoder is significantly costly to allow real-time learning. Figure4a shows the breakdown of ManiHD and baseline HD energy consumption. The results are reported when both approaches are providing the same classification accuracy. The baseline HD computing uses $D = 10k$ to provide high classification accuracy, while ManiHD uses $D = 4k$ to achieve the same quality of classification. This lower dimensionality significantly improves ManiHD efficiency during the training/inference phases, since the costly similarity search performs with higher efficiency. As Figure 4 shows, in ManiHD, the encoding module takes 72% of the training phase, while this portion is less than 20% on the baseline HD.

In order to run ManiHD on small and tiny embedded devices, we need to reduce overall energy consumption. This means that we need to reduce the computation cost of the encoding module. To this end, we proposed the idea of using a quantized projection matrix, where every element can be represented using n-bits ($n << 32$). Figure 4b shows the quality loss of ManiHD for activity recognition application [25] when the precision of elements in the projection matrix reduces from 32-bits. The right y-axis in Figure 4b shows the normalized energy consumption of ManiHD encoding during different bit precisions. The energy results are reported for FPGA. Since FPGAs have limited DSP resources, they can significantly speedup the encoding computation when quantization reduces to fewer bits. This is because FPGAs can use their highly parallel lookup table resources to parallelize the computation. However, the lower precision projection matrix results in a quality loss. For example, quantizing the projection matrix from 32-bits to 2-bits results in 3.4% quality loss while improving the energy efficiency by 5.8× times.

We observe that encoding data into higher dimensionality can compensate for the quality loss from quantization. Even with increasing dimensionality, the new quantized encoder has much lower computation cost on FPGA. This is because FPGAs have enough parallelism/resources for low-precision computations. In addition, the increase in hypervector dimensionality has a negative effect on training and associative search block. To provide maximum efficiency, we have optimized the system (devices dimensionality and level of quantization) such that the encoding and associative search consume the same amount of energy.

TABLE I
DATASETS (n: FEATURE SIZE, K: NUMBER OF CLASSES)

	n	K	Data Size	Train Size	Test Size	Description
MNIST	784	10	220MB	60,000	10,000	Handwritten Recognition[27]
PECAN	312	3	34MB	22,290	5,574	Smart home power prediction [28]
SPEECH	617	26	19MB	6,238	1,559	Voice recognition [28], [29]
UCIHAR	561	12	10MB	6,213	1,554	Activity recognition(Mobile)[25]
EXTRA	225	4	140MB	146,869	16,343	Phone position recognition[30]

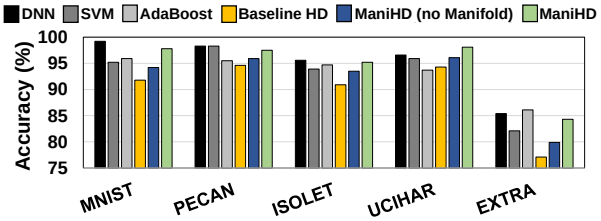

Fig. 5. ManiHD classification accuracy vs. the state-of-the-art algorithms.

V. EVALUATION

A. Experimental Setup

The proposed ManiHD framework has been implemented with both software and hardware support. In software, we design ManiHD in C++ for model training and verification while exploiting Scikit-learn library [26] for manifold learning. In hardware, we used a similar framework as [9] to implement ManiHD on Kintex-7 FPGA. The FPGA design accelerates the encoding and learning procedure by parallelizing element-wise computations for hypervectors. We test ManiHD efficiency and accuracy on a wide range of classification datasets. Table I summarizes the details of the datasets.

B. ManiHD Classification Accuracy

Figure 5 compares the classification accuracy with state-of-the-art classification algorithms, including Deep Neural Network (DNN), Support Vector Machine (SVM), and AdaBoost. We also compare the accuracy of ManiHD with a state-of-the-art HD-based classifier published in [19], which uses a linear encoding method, as the baseline. The results are reported when all algorithms are performing in a central node that considers all features given in the dataset. The DNN models are trained with Tensorflow [31], and we exploited the Scikit-learn library [26] for the other algorithms. We exploit the common practice of the grid search to identify the best hyper-parameters for each model. The accuracy of ManiHD is reported for $D = 4000$ dimensions. Our evaluation shows that ManiHD provides comparable classification accuracy to the sophisticated non-HD algorithms. As compared to the baseline HD computing, ManiHD can achieve, on average, 4.7% higher classification accuracy, since our new encoding method non-linearly maps the data to the high dimensional space whereas the baseline HD encoding linearly performs the encoding. ManiHD enhanced with manifold can further improve the classification accuracy, as it can identify better features before mapping them into high-dimensional space. Our evaluation shows that ManiHD can provide, on average, 2.6% (4.8%) higher classification accuracy as compared to ManiHD without manifold (the baseline HD computing).

C. ManiHD Accuracy-Efficiency

ManiHD efficiency and accuracy depend on a portion of the training data used for the manifold. From an accuracy point

TABLE II
THE IMPACT OF MANIFOLD DATA ON MANIHD EFFICIENCY

	0	*0.1%*	*0.2%*	***0.5%***	*1%*	*1.5%*	*1%*
Dimensions (D)	10k	7.5k	5.5k	**4k**	3.5k	3.5k	3k
Manifold Overhead	0.00	0.06	0.10	**0.46**	0.95	1.12	1.58
Training Speedup	1	1.07	1.32	**1.52**	1.34	1.04	0.86
Inference Speedup	1	1.06	1.38	**1.81**	2.07	2.01	2.31

of view, using a larger portion of data for manifold results in improving the classification accuracy. However, it comes with the overhead of processing manifold data. Note that the manifold overhead is only on the training phase, while the inference can get the advantage of the manifold for higher classification accuracy. Here, we perform an experiment to show the trade-off in selecting a suitable portion of manifold learning. Table II shows the impact of partial data training of manifold in ManiHD accuracy and efficiency. All results are reported when ManiHD provides the same accuracy.

Increasing a portion of manifold data results in providing higher classification accuracy. This enables us to reduce ManiHD dimensionality, as manifold already combines useful features in original data. As Table II reports, this dimension reduction can result in more efficient training and inference phases. However, learning the manifold by itself adds an extra cost to the ManiHD training phase. This cost depends on the portion of data used to learn manifold (quadratic relation). To design an efficient system, we need to ensure that the manifold's overhead does not overcome the efficiency coming from reducing dimensionality. Our evaluation shows that using 0.5% of data for learning manifold results in reducing the hypervector dimensionality by half. This lower dimensionality not only compensates the overhead of manifold but also results in 1.52× and 1.81× improvement in the training and inference performance, respectively. Further increasing the manifold data results in saturation in the classification accuracy. In other words, at the same level of accuracy, ManiHD provides a small reduction in the dimensionality. In addition, the overhead of the manifold increases quadratically with the number of data points. As results in Table II shows, using 1.5% of data for manifold results in a 14% slower training process. Note that the overhead of learning manifold is not on the inference phase. For systems optimized only for inference, a larger portion of data used to learning manifold results in higher accuracy or potentially lower dimensionality.

D. ManiHD Efficiency

We compare the computation efficiency of the DNN and HD computing algorithms. Figure 6 compares the efficiency of the training and inference procedure for the different configurations. All results are normalized to the execution time and energy consumption of DNN. All designs run on the same FPGA platform. We used DNNWeaver V2.0 [32] for efficient implementation of the DNN inference, and FPDeep [33] for NN training on a single FPGA device. FPGA implementations are optimized to maximize performance by utilizing FPGA resources. All HD-based approaches are compared when they provide the same classification accuracy. Therefore, when ManiHD with manifold works in $D = 4k$ binary, ManiHD without manifold and the baseline HD work in $D = 10k$ binary and non-binary representations. All results listed in Figure 6

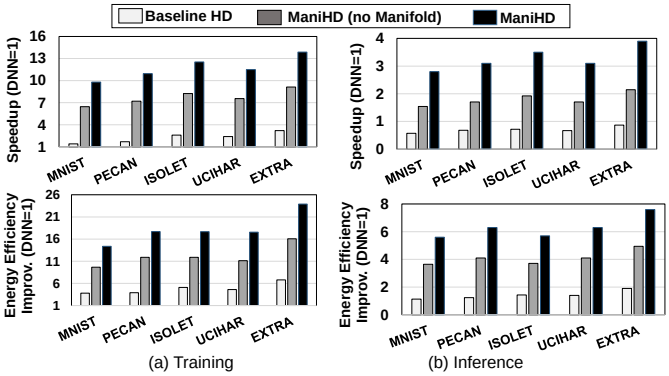

Fig. 6. ManiHD efficiency during training and inference phases.

are relative to DNN performance and energy efficiency. During training, ManiHD with manifold (without manifold) achieves, on average, 12.3× (7.7×) faster and 19.3× (12.0×) more energy-efficient computation as compared to FPGA-based DNN implementation, respectively. The high efficiency of ManiHD in training comes from: (i) ManiHD capability in creating an initial model that significantly lowers the number of required retraining iterations. (ii) It eliminates the costly gradient descent for the model update. In addition, ManiHD provides 5.4× faster and 3.8× higher energy efficiency as compared to the baseline HD computing. This efficiency comes from (i) ManiHD advance encoding method that enables ManiHD to provide maximum accuracy in 0.4 × lower dimensionality, (ii) ManiHD capability to perform the majority of training/inference over the binary model.

Figure 6b also compares ManiHD inference efficiency with DNN and the baseline HD. Although manifold does not affect the inference efficiency, it enables ManiHD to work on lower dimensionality, resulting in a higher computation efficiency. Our evaluation shows that ManiHD with manifold provides 3.2× and 6.3× (4.4× and 4.6×) faster and more efficient inference as compared to DNN (baseline HD computing).

E. Online Manifold Learning

As we discussed in Section IV-A, in real systems, data points are highly correlated and have high temporal locality. This means that the data points are dynamically changing during time. Therefore, ManiHD with a pre-trained manifold cannot provide suitable accuracy. Here, we design an experiment to show the advantage of ManiHD for online learning. We enable online learning on the smart home project, where the usage of electricity changes during different seasons. We perform two experiments over this dataset to show the impact of the adaptive manifold update on ManiHD encoding: (i) we train ManiHD on the training data (collected from summer season) and learned manifold by sampling 4% of training data. (ii) we train ManiHD on the same training data, but we frequently update the manifold data during the inference. In each season, we sample 1% of data points and update the manifold and ManiHD encoder accordingly. Note that both static and adaptive manifold updates have the same computational overhead as they process a total of 4% of data. Table III reports the classification accuracy of ManiHD in these two configurations during different time steps. Our evaluation shows that ManiHD using static manifold

TABLE III
IMPACT OF STATIC AND ADAPTIVE MANIFOLD LEARNING ON MANIHD CLASSIFICATION ACCURACY

	Summer	*Fall*	*Winter*	*Spring*	*AVERAGE*
Static Design	95.2%	92.7%	84.7%	86.1%	89.7%
Adaptive Design	94.4%	95.2%	93.6%	92.6%	94.0%

provides high classification accuracy in summer, while in other seasons, the classification accuracy significantly drops. This is because the manifold has been learned using summer data. However, ManiHD updating the manifold adaptively results in providing high accuracy in every season. Our evaluation shows that ManiHD using adaptive manifold provides 4.3% higher accuracy as compared to ManiHD with the static manifold.

F. ManiHD Encoding Quantization

As we discussed in Section IV-B, the ManiHD encoding module dominates the entire training/inference cost. In order to reduce the overall ManiHD efficiency, we propose the idea of encoding quantization. Our approach represents each element of the projection matrix with n-bits, where $n << 32$. However, quantization reduces ManiHD classification accuracy. ManiHD can compensate for the quality loss caused by quantization with increasing the hypervector dimensions. Figure 7 shows the number of required ManiHD dimensionality during different levels of encoder quantizations. As this graph shows, with no quantization provides maximum accuracy using $D = 4k$ while quantizing ManiHD encoder into 2-bits (3-bits) precision results in providing the maximum accuracy with $D = 7.1k$ ($D = 5.7k$) dimensions.

Although quantizing the projection matrix reduces the encoding cost, it increases dimensionality, directly impacting training, and inference cost. Figure 7 shows the breakdown of ManiHD energy consumption during the training phase. The energy values are reported for the encoding and training modules. As Figure 7 shows, the encoding module takes 72% the energy consumption over baseline ManiHD (Non-quantized). Quantizing the encoder to lower precision increases the training cost (higher dimensionality) while reducing the encoding cost. The optimized system for learning (encoding and training phase) provides maximum efficiency. This optimization depends on the underlying hardware. Since we select FPGA as hardware implementation, our goal is to select the best quantization that minimizes the total FPGA energy, which includes both encoding and training. Our evaluation shows that 3-bits quantization balances the energy consumption of the encoding and training phase, resulting in maximum efficiency.

VI. CONCLUSION

In this paper, we proposed ManiHD that supports adaptive and trainable encoder for efficient learning in high-dimensional space. ManiHD explicitly considers non-linear interactions between the features during the encoding. This enables ManiHD to provide maximum learning accuracy in much lower dimensionality. ManiHD also enables online learning by sampling data points and capturing the essential features in an unsupervised manner. Our evaluation shows that ManiHD provides, on average, 12.3× faster and 19.3× more energy-efficient training as compared to state-of-the-art learning algorithms.

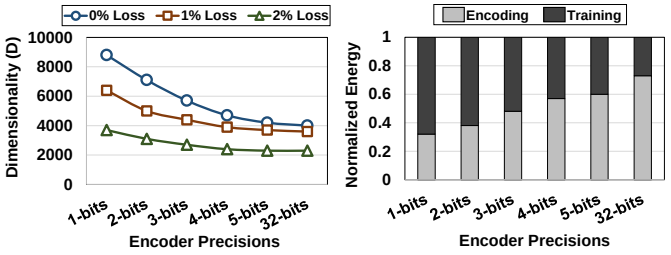

Fig. 7. Impact of the encoder quantization on (a) ManiHD dimensionality that provides a certain accuracy, (b) breakdown of the energy consumption in the encoding and training phases.

ACKNOWLEDGMENT

This work was partially supported by Semiconductor Research Corporation (SRC) Task No. 2988.001, National Science Foundation grant #2019511, and Louisiana Board of Regents Support Fund LEQSF(2020-23)-RD-A-26. Mohsen Imani and Yeseong Kim are co-corresponding authors of the paper.

REFERENCES

[1] M. Shafique and Nothers, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *D&T*, vol. 37, no. 2, pp. 30–57, 2020.
[2] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*, pp. 802–815, IEEE, 2019.
[3] J. Choi *et al.*, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.
[4] 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.
[5] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *CLOUD*, pp. 435–446, IEEE, 2019.
[6] M. Imani *et al.*, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *DAC*, pp. 1–6, 2019.
[7] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, ACM, 2016.
[8] M. Imani *et al.*, "Quanthd: A quantization framework for hyperdimensional computing," *TCAD*, 2019.
[9] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, pp. 53–62, 2019.
[10] S. Salamat *et al.*, "Accelerating hyperdimensional computing on fpgas by exploiting computational reuse," *TC*, 2020.
[11] Y. Kim *et al.*, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *DATE*, IEEE, 2020.
[12] B. Khaleghi *et al.*, "Prive-hd: Privacy-preserved hyperdimensional computing," *arXiv preprint arXiv:2005.06716*, 2020.
[13] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *HPCA*, IEEE, 2021.
[14] M. Imani *et al.*, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *FCCM*, pp. 190–198, IEEE, 2019.
[15] B. Khaleghi *et al.*, "tiny-HD: Ultra-Efficient Hyperdimensional Computing Engine for IoT Applications," in *DATE*, IEEE, 2021.
[16] S. Gupta *et al.*, "Thrifty: Training with hyperdimensional computing across flash hierarchy," in *ICCAD*, pp. 1–9, IEEE, 2020.
[17] A. Moin *et al.*, "Analysis of contraction effort level in emg-based gesture recognition using hyperdimensional computing," in *BioCAS*, pp. 1–4, IEEE, 2019.
[18] A. Mitrokhin *et al.*, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, no. 30, p. eaaw6736, 2019.
[19] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, p. 108, ACM, 2018.
[20] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS*, pp. 1177–1184, 2008.
[21] B. Schölkopf, "The kernel trick for distances," in *NIPS*, pp. 301–307, 2001.
[22] R. Abraham *et al.*, *Manifolds, tensor analysis, and applications*, vol. 75. Springer Science & Business Media, 2012.
[23] Y. Zhao *et al.*, "Multi-view manifold learning with locality alignment," *Pattern Recognition*, vol. 78, pp. 154–166, 2018.
[24] Z. Zhang *et al.*, "M-isomap: Orthogonal constrained marginal isomap for nonlinear dimensionality reduction," *SMC*, vol. 43, no. 1, pp. 180–191, 2012.
[25] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *AAL*, pp. 216–223, Springer, 2012.
[26] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *JMLR*, vol. 12, no. Oct, pp. 2825–2830, 2011.
[27] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[28] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.
[29] M. S. Razlighi *et al.*, "Looknn: Neural network with no multiplication," in *DATE*, pp. 1775–1780, IEEE, 2017.
[30] Y. Vaizman *et al.*, "Recognizing detailed human context in the wild from smartphones and smartwatches," *IEEE Pervasive Computing*, vol. 16, no. 4, pp. 62–74, 2017.
[31] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.
[32] H. Sharma *et al.*, "From high-level deep neural models to fpgas," in *MICRO*, p. 17, IEEE, 2016.
[33] T. Geng *et al.*, "Fpdeep: Acceleration and load balancing of cnn training on fpga clusters," in *FCCM*, pp. 81–84, IEEE, 2018.