SynergicLearning: Neural Network-Based Feature Extraction for Highly-Accurate Hyperdimensional Learning

Mahdi Nazemi
University of Southern California
mnazemi@usc.edu

Amirhossein Esmaili
University of Southern California
esmailid@usc.edu

Arash Fayyazi
University of Southern California
fayyazi@usc.edu

Massoud Pedram
University of Southern California
pedram@usc.edu

ABSTRACT

Machine learning models differ in terms of accuracy, computational/memory complexity, training time, and adaptability among other characteristics. For example, neural networks (NNs) are well-known for their high accuracy due to the quality of their automatic feature extraction while brain-inspired hyperdimensional (HD) learning models are famous for their quick training, computational efficiency, and adaptability. This work presents a hybrid, synergic machine learning model that excels at all the said characteristics and is suitable for incremental, on-line learning on a chip. The proposed model comprises an NN and a classifier. The NN acts as a feature extractor and is specifically trained to work well with the classifier that employs the HD computing framework. This work also presents a parameterized hardware implementation of the said feature extraction and classification components while introducing a compiler that maps any arbitrary NN and/or classifier to the aforementioned hardware. The proposed hybrid machine learning model has the same level of accuracy (i.e. ±1%) as NNs while achieving at least 10% improvement in accuracy compared to HD learning models. Additionally, the end-to-end hardware realization of the hybrid model improves power efficiency by 1.60x compared to state-of-the-art, high-performance HD learning implementations while improving latency by 2.13x. These results have profound implications for the application of such synergic models in challenging cognitive tasks.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Bio-inspired approaches**; **Supervised learning**; • **Computer systems organization** → **Architectures**.

KEYWORDS

Neural Networks, Hyperdimensional Learning, On-line Learning, Parallel Processing

1 INTRODUCTION

Machine learning models have proven successful in solving a wide variety of challenging problems such as computer vision and speech recognition. They are commonly characterized by their level of accuracy, computational/memory complexity, training time, and adaptability among other features. One can categorize machine learning models according to the aforesaid characteristics. For example, neural networks (NNs) typically achieve high accuracy [1], are computationally expensive [2], have long training times [3], and tend to forget previously learned information upon learning new information (aka catastrophic forgetting) [4–6]. A machine learning model is more viable for on-chip learning (also called learning on-a-chip which refers to designing a custom chip that can be used for both training and inference) when it has low computational/memory complexity and supports one-pass training/fine-tuning while maintaining a high level of accuracy.

The main reason behind the high accuracy of NNs is their ability to automatically extract high-quality, high-level features from labeled data. AlexNet [7] is an outstanding example that clearly demonstrates the gap between the quality of features extracted by NNs compared to handcrafted features extracted by experts in the domain (in the ImageNet Large Scale Visual Recognition Challenge [8], AlexNet was able to achieve 10.8% higher accuracy compared to the runner up, which used handcrafted features). Unfortunately, the high accuracy of NNs is accompanied by an enormous computational/memory cost during training and inference. Training an NN is a time-consuming, iterative process where in each iteration, all training data is applied to the model and the parameters of the model are updated according to stochastic gradient descent.

As another example, hyperdimensional (HD) learning models train quickly, are highly adaptable and computationally efficient (compared to NNs), but suffer from lower levels of accuracy compared to NNs [9]. HD learning uses randomly generated, high-dimensional vectors to project training data into HD space such that samples belonging to the same class are placed in close proximity of each other, forming a cluster in the HD space. It then defines HD centroids that represent different classes. This relatively simple training process only requires one pass over the training data. It also enables efficient incremental, lifelong learning because updating the model with new training data is as simple as updating the cluster centroids. The major disadvantage of HD learning is that it

works with raw or handcrafted input features, which are inferior to the ones extracted by NNs.

The complementary characteristics of NNs and HD models encourage the introduction of a hybrid, synergic machine learning model that builds on their strengths while avoiding their shortcomings. However, simply employing NNs for feature extraction and HD models for classification so as to enable on-chip learning has the following challenges. Not only is the training of NNs for feature extraction an iterative, energy-consuming process but also it requires access to both previous training data and newly provided data to avoid catastrophic forgetting. Therefore, frequent weight updates of NNs can be extremely costly in the context of learning on-a-chip. Additionally, the HD learning models that work well for solving cognitive tasks have a huge number of dimensions, e.g., 10,000, which requires their hardware implementation to time-share resources and therefore, have a relatively high latency. This prevents real-time fine-tuning of the model when new training data becomes available. Moreover, training NNs for feature extraction separately from the design of the HD learning model produces suboptimal results because it does not account for the effect of HD classification layers on the NN feature extraction layers and vice versa. This means that the prediction/classification accuracy of the overall hybrid solution will suffer.

This work presents SynergicLearning, a hybrid learning framework for incremental, on-line learning on a chip. SynergicLearning is comprised of three components which enable end-to-end learning:

(1) **A Two-step Training Approach:** This training approach first trains an NN while including some components of the HD learning system in the NN's training loop to learn high-quality, high-level features that are specifically tailored for the HD learning system. It then passes training data (including the initial data as well as the ones that are generated during the lifetime of the model) through the feature extraction layers of the NN to provide features for training/fine-tuning of the HD classifier (the neural network parameters are fixed at this step). Such a two-level training approach enables automatic feature extraction while reducing the number of dimensions in the HD classifier by two to three orders of magnitude[1].

(2) **An On-chip Learning Module:** This module is comprised of parameterized NN and HD processing modules, which respectively execute operations required by the NN feature extraction layers and operations required by the HD classifier. The NN processing module includes a systolic array which performs vector-matrix multiplications and an ALU which supports operations such as batch normalization, pooling, and ReLU. The HD processing module supports the arithmetic operations defined in the HD computing including *binding*, *bundling*, and distance calculation (Section 2 details these operations). The parameterized hardware implementation enables efficient exploration of the design space to find configurations that satisfy the design constraints such as energy and resource utilization.

(3) **A Compiler:** The custom compiler performs code optimizations and generates instructions that efficiently schedule different operations required by the NN feature extraction

---

[1]While the term *hyperdimensional* learning is no longer applicable to such a classifier, we keep using the same term to highlight the fact that the operations used in the classifier are based on those defined in the hyperdimensional computing framework.

and HD classification steps (e.g., vector-matrix multiplications and data movement) on the target platform.

Table 1 compares different characteristics of NNs, HD learning systems (HDL), and the proposed SynergicLearning approach. It is observed that SynergicLearning enjoys automatic feature extraction and high accuracy because it employs an NN that is tailored for HDL. Furthermore, it only requires one pass to train/fine-tune its HD classifier and last but not least, it does not require accessing previous training samples to update the model when new data becomes available.

The remainder of this paper is organized as follows. Section 2 explains the preliminaries on HD computing, discusses some of its shortcomings, and motivates the presented solution. Next, Section 3 details the proposed learning framework while Section 4 explains the proposed hardware architecture and compiler for inference. After that, Section 5 presents the experimental results while Section 6 briefly reviews the related work on HD computing. Finally, Section 7 concludes the paper.

2 PRELIMINARIES & MOTIVATION

HD computing defines a new computation framework that relies on high-dimensional random vectors (aka *hypervectors*) and the arithmetic operations that manipulate such large random patterns. An HD system starts by randomly generating d^h-dimensional, holistic seed hypervectors with independent and identically distributed (i.i.d) elements. This means that the information encoded into each hypervector is uniformly distributed over all its elements. Therefore, unlike the conventional computing framework, elements in different bit positions in hypervectors are equally significant. The seed hypervectors are typically stored in a memory called the *cleanup memory*. The arithmetic operations defined on the seed hypervectors, e.g. binding and bundling, enable meaningful computations in the corresponding hyperspace. The focus of this paper is on binary hypervectors where each element is equally likely to be a zero or one. Binary hypervectors enjoy simplified, hardware-friendly arithmetic operations.

The distance between two binary hypervectors is measured in normalized Hamming distance, i.e. the number of bit positions where the values of hypervectors differ, divided by d^h. Consequently, the distance is always in the range zero to one inclusive. However, because the distance between two randomly generated hypervectors follows a binomial distribution, most hypervectors are about 0.5 apart from one another (when d^h is large) and therefore, are nearly orthogonal (aka *unrelated*). Additionally, flipping the values of a relatively large portion of elements in a hypervector, e.g. one-third of all elements, results in a hypervector that is closer to the original hypervector compared to its unrelated hypervectors. This results in considerable tolerance to noise and approximation. When the cleanup memory is queried with a noisy hypervector, it returns the seed hypervector that is closest to the input query, hence the name cleanup.

Two of the commonly used arithmetic operations in HD computing are binding and bundling. The binding operation is used for variable-value association. Assume variable z and its corresponding value z_0 are represented with unrelated hypervectors z and z_0, respectively. Then, the bound pair $z = z_0$ can be represented by $z * z_0$, where element-wise multiplication ($*$) is replaced with element-wise XOR for binary hepervectors. The resulting hypervector is unrelated to both z and z_0. However, each original hypervector

Table 1: Comparison of different characteristics of NNs, HD learning systems, and SynergicLearning.

Machine Learning Model	Automatic Feature Extraction	High Accuracy	One-pass Training/Fine-tuning	Adaptable w/o Accessing Previous Training Samples
NN	✓	✓	✗	✗
HDL	✗	✗	✓	✓
SynergicLearning	✓	✓	✓	✓

can be recovered from the resulting hypervector given the other, e.g. $z_0 = S((z * z_0) * z)$, where $S(.)$ looks up the cleanup memory. This process is called *unbinding*. The bundling operation condenses a list of hypervectors into a single representative hypervector that is similar to all its constituents. This is achieved by summing up all hypervectors, followed by the comparison of each element in the resulting (summation) hypervector with half the number of original hypervectors to create a binary hypervector. If the original hypervectors are bound, their variables and/or values can be found through unbinding the bundled hypervector.

The HD computing framework can be used to solve cognitive tasks such as speech recognition and activity recognition [9–11]. Alg. 1 summarizes different steps of training an HD model. The inputs to the algorithm are d^l-dimensional input features, their corresponding labels/classes, the dimension of hypervectors (d^h), and the number of quantization levels used to discretize the input values while the outputs are the HD centroids representing each class. The training starts with the generation of seed hypervectors for all d^l features as well as the quantized values they can assume. While the seed hypervectors for features are generated randomly, the ones for quantized values are found by randomly flipping a specific number of bits of a seed hypervector to ensure the similarity of the hypervectors representing nearby values. Next, each feature and its value are bound and the set of all bound hypervectors are bundled into a single hypervector (aka *encoding*). Finally, the encoded hypervectors are categorized according to their labels and the set of hypervectors belonging to a class are bundled to find a representative centroid. During inference, the closest centroid to an encoded test sample (in terms of normalized Hamming distance) determines the model's prediction.

By fixing d^h and increasing q, the hypervectors representing different quantization levels become more similar because fewer bits are flipped across consecutive hypervectors. This, in turn, complicates the unbinding process because the cleanup memory may return the wrong values. Fig. 1 clearly illustrates this phenomenon by depicting the mean and standard deviation of the normalized absolute error between the input features and the decoded features of their encoded hypervectors. Ideally, decoding encoded hypervectors should return the exact same low-dimensional features as the original inputs (i.e., zero error), but this does not happen in practice. We believe this phenomenon is the main reason for the relatively poor performance of HD models compared to some other machine learning models such as NNs. **Therefore, creating input features that are aware of the error due to very similar quantization levels can improve classification accuracy significantly, especially at lower d^h.**

3 PROPOSED METHOD

Fig. 2 demonstrates a high-level overview of the proposed hybrid learning framework. The proposed framework comprises two major components: an encoder-aware NN for high-quality feature extraction and an HD classifier.


Algorithm 1 Training an HD Model

Input:

$X_{n \times d^l} = x_{1..n}, x_i \in \mathbb{R}^{d^l}$ //the low-dimensional input features
$y = y_{1..n}, 1 \le y_i \le c$ //the target labels/classes
d^h //the number of hyperspace dimensions
q //the number of quantization levels

Output:

$T_{c \times d^h} = t_{1..c}$ //the HD centroids
1: generate $S_{d^l \times d^h} = s_{1..d^l}$ //seed hypervectors for features
2: $p = \lfloor \frac{d^h}{q} \rfloor$ //number of bits to flip
3: generate q_1 randomly
4: **for** $i = 2..q$ **do**
5: $\quad q_i =$ randomly pick p unflipped bits and flip them in q_{i-1}
6: **end for**
7: $Q_{q \times d^h} = q_{1..q}$ //seed hypervectors for levels
8: **for each** x_i **do** //**encode all samples**
9: $\quad x_i^q =$ quantize(x_i, q) //quantize real values to integers
10: $\quad X_i = \emptyset$
11: $\quad$ **for** j in $1..d^l$ **do** //**bind feature-value pairs**
12: $\quad\quad X_i = X_i \cup$ bind($s_j, q_{x_{i_j}^q}$)
13: $\quad$ **end for**
14: $\quad x_i^{enc} =$ bundle(X_i) //**bundle bound hypervectors**
15: **end for**
16: $T_1 = T_2 = ... = T_c = \emptyset$
17: **for each** x_i^{enc} **do** //**group encoded inputs by labels**
18: $\quad T_{y_i} = T_{y_i} \cup x_i^{enc}$
19: **end for**
20: **for each** T_k **do** //**bundle all members of each class**
21: $\quad t_k =$ bundle(T_k)
22: **end for**
23: return T

The NN includes feature extraction layers, an HD encoder-decoder pair (i.e. HD codec), and classifier layer(s). It takes the input features, passes them through the said components (aka forward propagation), and calculates a loss value by comparing the predicted labels with the expected ones. It then updates the model parameters, i.e. weights and biases, by backpropagating the loss value using the derivative of the operations defined in the NN. Because the operations defined in the codec are not differentiable and the fact that an ideal codec should behave like the identity function, the codec's derivative is approximated with that of the identity function during backpropagation.

Pre-processing input features with an NN has numerous important advantages. First, **including the codec in the training loop encourages the NN to adjust its parameters such that it minimizes the impact of the codec's error on classification accuracy.** Training two identically initialized NNs, one including a codec and the other without a codec, would result in a completely different set of parameters. Second, the number of features
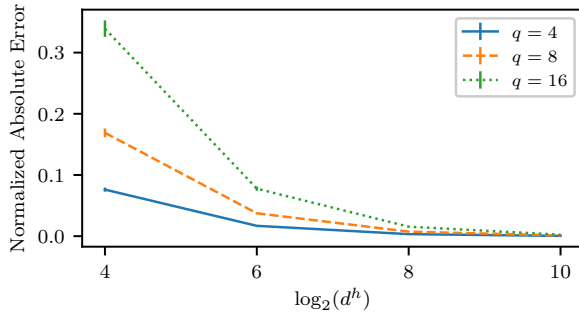
Figure 1: The mean and standard deviation of the normalized absolute error between the input features and the decoded features of their encoded hypervectors for different values of d^h and q. Ideally, this error should be zero everywhere. However, the error has a non-zero value even at extremely high dimensions ($d^h \simeq 10,000$).

extracted by the NN (d^{NN}) can be much lower than the number of low-dimensional input features (d^l), which in turn reduces the complexity of the HD classifier. Third, because the NN extracts encoder-aware features, d^h can be reduced by two to three orders of magnitude compared to the existing HD systems. In other words, because the degree of similarity of hypervectors representing quantization levels is less concerning, lower d^h values can work equally well. Fourth, there is a large body of work on reducing the complexity of NNs through quantization [12, 13], pruning [14], and knowledge distillation [15], to name but a few. This allows training NNs that are lightweight, thereby adding little overhead to the overall hardware cost.

The HD classifier is very similar to the one described in Alg. 1. The only difference is that it takes the output of NN's feature extraction layers (x_i^{NN}) instead of the original input features (x_i). Therefore, it not only benefits from the inherent strength of NNs in feature extraction but also enjoys features that are specifically tailored for the HD encoder.

4 PROPOSED HARDWARE ARCHITECTURE & COMPILER

The proposed hardware architecture implements an end-to-end, fully-parameterized implementation of SynergicLearning for inference. It consists of two major hardware components: an NN processing module, which includes a systolic array and an ALU, and a fully-parallel HD processing module which supports various operations such as binding, bundling, and distance calculation.

4.1 NN Processing Module

Fig. 3 demonstrates a high-level overview of the NN processing module, which comprises the following components:

- the systolic array, which consists of a two-dimensional array of processing elements,
- on-chip memories (i.e. weight, input, and output buffers), which act as an intermediate storage between the DRAM and the systolic array,
- tree adders, each of which performs a summation over a row of the systolic array, and
- ALUs, which support activation functions, batch normalization, pooling, etc.

Processing each layer of the NN requires the following operations. First, the weights are read from the external memory (DRAM) and stored in the weight buffer while inputs are read either from the DRAM or the output buffer. Next, the systolic array and tree adders calculate the neurons' pre-activation values by implementing vector-matrix multiplications. Then, ALUs apply batch normalization, activation function, pooling, etc. to pre-activations to generate the output features. Finally, the output features are either rerouted to the input buffers or written back to the DRAM.

The systolic array implements a weight-stationary dataflow [2], which reuses each weight value in different computations involved in vector-matrix multiplication and therefore, reduces the overhead associated with data movement. In this dataflow, the number of cycles it takes to process each layer of the NN is approximated by

$$(\lceil \frac{d_{l_i}}{w_{sys}} \rceil + \log_2 w_{sys}) \times \lceil \frac{d_{l_{i+1}}}{h_{sys}} \rceil,$$

where d_{l_i} is number of neurons in the ith layer and w_{sys} (h_{sys}) is the number of columns (rows) in the systolic array. $\log_2 w_{sys}$ represents the depth of each tree adder.

For mapping a neural network presented in high-level languages to the target FPGA, we developed an in-house compiler called SynergicCompiler. Since all hardware designs presented in this paper perform the same computation, i.e. a three-level nested loop for fully connected layers, the space explorations are defined by transformations (e.g., block, reorder, and parallelize) on the nested loop. Therefore, the compiler tries various choices of loop ordering and hardware parallelism for computing these nested loops of NNs and finds the most efficient one in terms of latency. The SynergicCompiler also generates a static schedule for the data movements between hierarchies of memories, e.g., between external memories and buffers, and buffers and registers within PEs. Static scheduling mitigates the need for complex handshaking and improves the scalability and performance of the processing modules. Finally, the compiler delivers a set of instructions that efficiently schedule different operations such as vector-matrix multiplications and data movement on the target platform. More details about the compiler are not included in this paper for brevity.

4.2 HD Processing Module

Fig. 4 demonstrates a high-level overview of the pipelined HD processing module, which comprises the following components:

- lookup tables (LUTs) that store hypervectors representing quantized levels,
- binding/unbinding units, which perform parallel XOR operations,
- majority counters, which compute the population count of a bit vector by incrementing a ($\log_2(d^l + 1) + 1$)-bit counter when a set bit is encountered and decrementing it when a reset bit is seen,
- comparators, which produce binary hypervectors from integer hypervectors, and
- tree adders and tree comparators, which implement a fully-parallel Hamming distance calculation and therefore, produce outputs in constant time.

The architecture of the proposed HD processing module has the lowest achievable latency but suffers from high resource consumption at large d^h values compared to other possible architectures
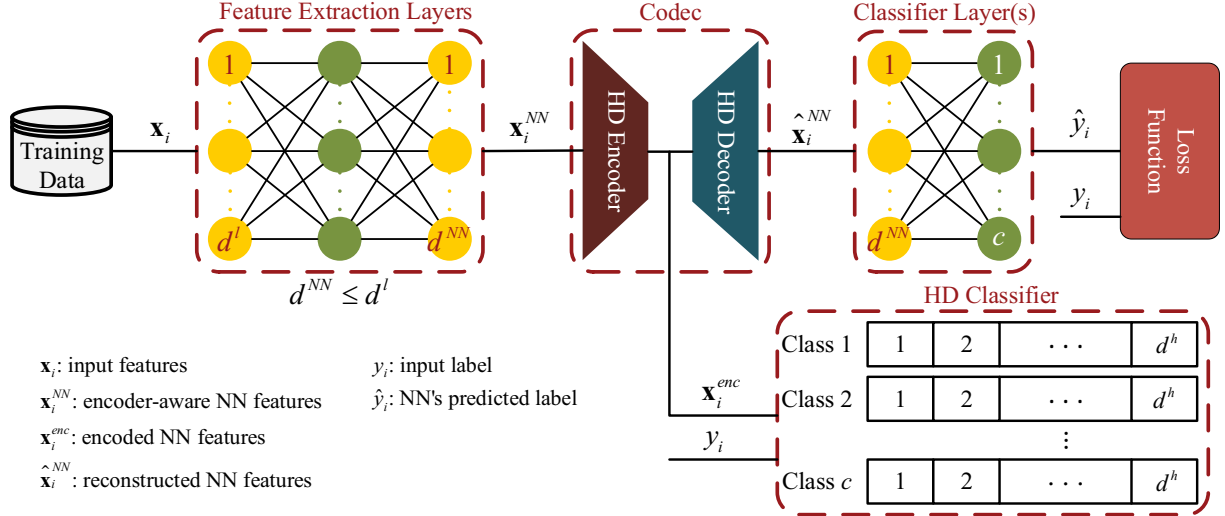
Figure 2: A high-level overview of the SynergicLearning framework. First, an encoder-aware NN is trained to extract high-quality, high-level features (top row of the figure). Next, encoded NN features are provided to train an HD classifier. Finally, during inference, the feature extraction layers of the NN and the HD classifier are both utilized to predict each test sample's label.
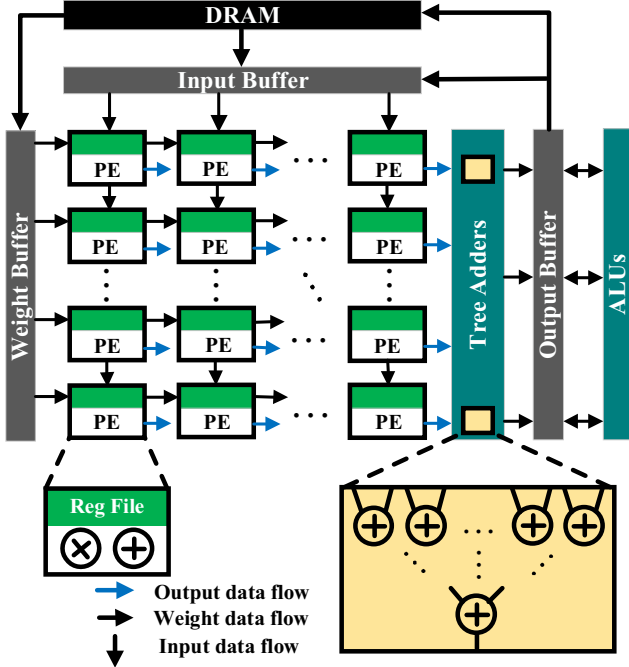

Figure 3: Architectural view of the NN processing module which includes a systolic array, on-chip memories, tree adders, and ALUs.

such as the ones explained in [16]. However, because SynergicLearning allows the utilization of HD learning systems with extremely low d^h values, the resource usage of the HD processing module will be negligible. Furthermore, because all the aforementioned components produce their results in constant time, the final output of the HD processing module will be produced in constant time too. Additionally, because of the pipelined implementation of the

HD processing module, it can produce an output every cycle, hence very high throughput.

5 RESULTS & DISCUSSION

5.1 Experimental Setup

5.1.1 Datasets. To study the effectiveness of SynergicLearning, we use two publicly available datasets: Human Activity Recognition (HAR) [17] and ISOLET [18]. HAR includes 10,299 samples, each of which contains 561 handcrafted features and a label that corresponds to one of six possible activities. ISOLET, on the other hand, contains 7,797 samples, each of which includes 617 handcrafted features and a label that corresponds to one of the 26 characters in the English alphabet. The goal is to take the input features and their labels and train classifiers that predict labels of unseen samples accurately.

5.1.2 Training Framework. We implement a PyTorch-compatible [19] HD computing library that includes operations such as binding/unbinding, bundling, encoding, and decoding. Because of the compatibility with PyTorch, the operations can be mapped efficiently to either CPUs or GPUs. Additionally, they can be easily integrated into existing PyTorch designs such as NNs.

We also implement a training ecosystem that takes a user-defined (possibly existing) NN architecture and the parameters of the HD learning system (e.g. d^h and q) and automatically glues different components together to enable encoder-aware training of the neural network. Similarly, it includes easy-to-use HD training modules. This training ecosystem allows us to quickly explore different designs and compare their accuracy.

5.1.3 Neural Network Training. We train all NNs by minimizing a cross-entropy loss function for 120 epochs, with a batch size of 256, and an l_2 regularizer. Additionally, we use a learning rate scheduler similar to the one described in [20] where the maximum learning rate is set to 0.01 while the number of steps per epoch is 25.
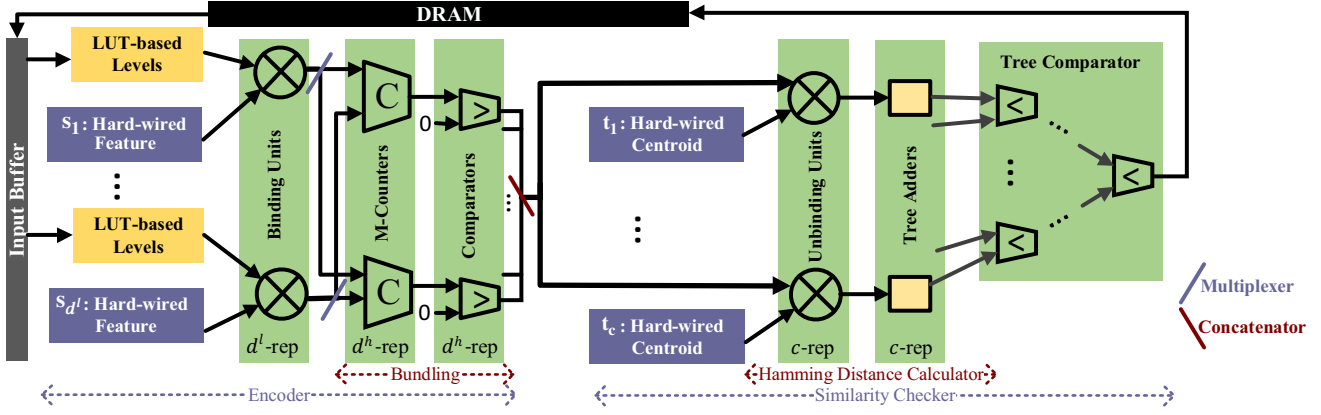
Figure 4: Architectural overview of the HD processing module which includes lookup tables that store hypervectors representing quantized levels, binding/unbinding units, majority counters, comparators, tree adders, and tree comparators.

5.1.4 **Hardware Emulation Framework**. To implement the NN and HD processing modules, we use the Xilinx SDAccel which provides a toolchain for programming and optimizing different applications on Xilinx FPGAs using a high-level language (C, C++ or OpenCL) and/or hardware description languages (VHDL, Verilog and SystemVerilog), as well as a runtime based on the OpenCL APIs that can be used by the host-side software to interact with the accelerator. We evaluate our proposed architecture using SDAccel on the ISOLET dataset targeting the Xilinx UltraScale+ VU9P FPGA on AWS EC2 F1 instances. We also use the Vivado power report provided by Xilinx to assess the power consumption of each design.

5.2 The Impact of NNs on the Quality of HD Features

In this section, we study the impact of NNs on the quality of encoded HD features by visualizing different samples of the HAR dataset in two-dimensional (2D) space. The feature extraction layers of the NNs consist of two fully-connected layers, each of which has 561 neurons. We deliberately keep the number of neurons in the final feature extraction layer the same as the one for the input features (i.e. $d^{NN} = d^l$) to ensure the difference across the results of various experiments is only due to the introduction of NNs. We use ReLU and PACT [12] for the activation functions of the first and second layer, respectively.

Fig. 5 shows the 2D representation of the encoded hypervectors of the test set for three different designs: HDL, NN followed by HDL, and encoder-aware NN followed by HDL (i.e. the proposed flow). To obtain the 2D representation, we employ t-distributed stochastic neighbor embedding (t-SNE) [21], which is a technique used for visualizing high-dimensional data. t-SNE tends to provide good visualizations because it tries to keep the similarities in HD space in the 2D representation as well. The 2D representations of hypervectors belonging to different classes are shown using different colors. For figures 5a-5c, we use $d^h = 16$, and for figures 5d-5f, we use $d^h = 10,240$. For all experiments, $q = 4$.

For small values of d^h (e.g. 16), it is observed that HDL performs poorly in the separation of points in the HD space (Fig. 5a). On the other hand, the addition of an NN to the flow helps with more proper separation of data points (Fig. 5b) while introducing an encoder-aware NN leads to a near-perfect clustering of data (Fig. 5c). The accuracy values reported in Fig. 5a-5c further support this

observation. For large values of d^h (e.g. 10,240), it is observed that HDL performs relatively well while models that include NNs still outperform the HDL model by a large margin (Fig. 5d-5f). In this configuration, the model that includes an NN and the one that has an encoder-aware NN perform almost equally well.

5.3 Comparison of Classification Accuracy

Table 2 compares the highest values of accuracy reported for NNs and HD learning systems with the proposed SynergicLearning approach on the HAR and ISOLET datasets. It is observed that on these datasets, the proposed hybrid model outperforms both NNs and HD learning systems used in the prior work.

Fig. 6 compares classification accuracy of three different models (HDL, NN followed by HDL, and encoder-aware NN followed by HDL) for different values of d^h and q on HAR and ISOLET datasets. It is observed that the model that includes an NN consistently outperforms HDL while the model that has an encoder-aware NN outperforms the other two in almost all experiments. On the HAR dataset, the difference between the model with an encoder-aware NN and the HDL model is as large as about 63% at $d^h = 16$ while it decreases to about 14% at $d^h = 10,240$. Similarly, On the ISOLET dataset, the difference between the model with an encoder-aware NN and the HDL model is as large as about 83% at $d^h = 16$ while it decreases to about 10% at $d^h = 10,240$.

Another key observation is that the model with an encoder-aware NN achieves almost the same level of accuracy at different values of d^h. This is particularly interesting from a hardware cost perspective, because we can pick the lowest value of d^h (16 in these experiments) and achieve significant reduction in resource utilization while maintaining high accuracy.

We also study the effect of different random seeds for initialization of NN weights and randomly generated seed hypervectors on classification accuracy. Based on our experiments, the difference between the lowest and highest values of classification accuracy across designs that use different seeds is at most 1%. We believe such variation in classification accuracy is acceptable.

---

[2]They also reported higher accuracy of 97.6 % when they added statistical features and data centering methods to their convolutional neural network.

(a) HDL   **(b) NN followed by HDL**   **(c) Encoder-aware NN followed by HDL**

(d) HDL   **(e) NN followed by HDL**   **(f) Encoder-aware NN followed by HDL**
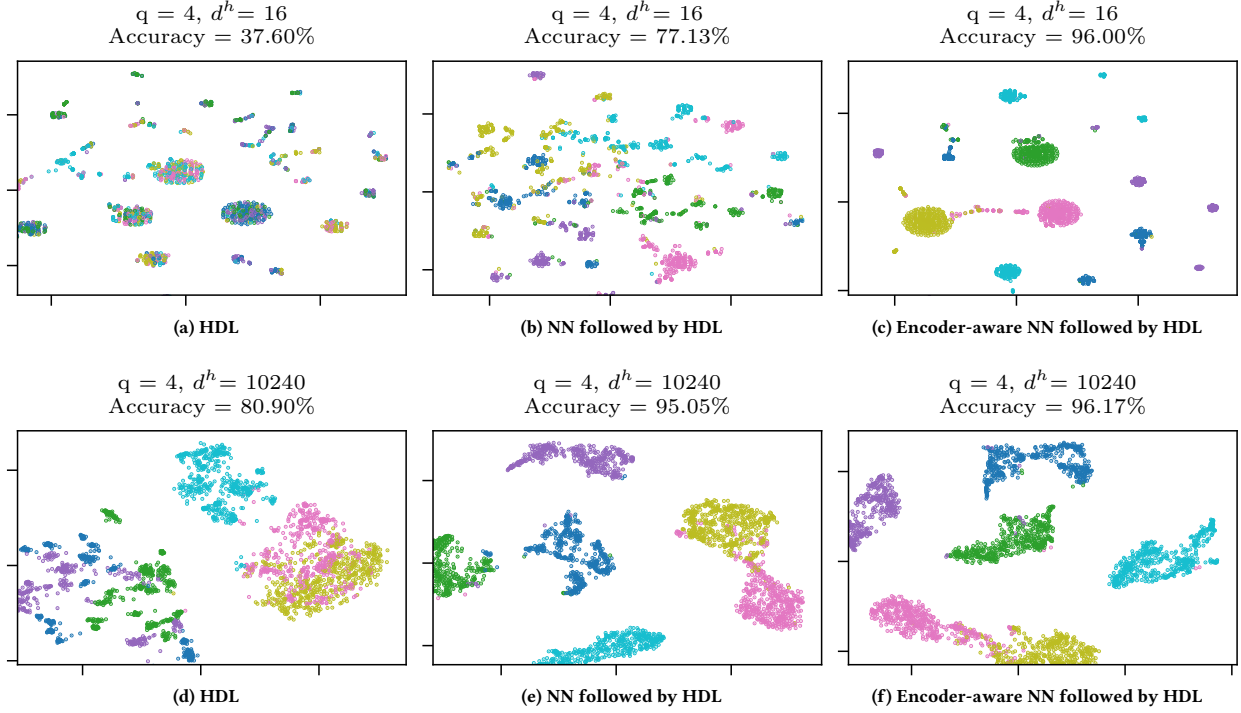
Figure 5: Two-dimensional (t-SNE) representation of the encoded hypervectors of the HAR dataset for three different designs: HDL, NN followed by HDL, and encoder-aware NN followed by HDL.

Table 2: Top accuracy reported for NNs, HD learning systems, and SynergicLearning on HAR and ISOLET datasets.

Dataset	Machine Learning Model	Accuracy (%)
HAR	NN [22] [‡2]	95.31 %
	HDL [23]	93.4%
	SynergicLearning	96.44 %
ISOLET	NN [24, 25]*	95.9 %
	HDL [25]	93.8 %
	SynergicLearning	96.67 %

[‡]Uses a convolutional neural network.

*Uses a fully-connected network with 48 hidden layers.

5.4 Incremental Learning

Table 3 compares the accuracy of HD learning models and SynergicLearning when a portion of data is initially used for training while the remaining data is used for fine-tuning the model on a chip. Because on-chip-learning is extremely costly for NNs, we do not consider them in this comparison. As expected, the HDL model is insensitive to whether the training data is provided incrementally or all at once and therefore, its accuracy remains constant

Table 3: Comparison of the effect of incremental learning on the accuracy of different models on the ISOLET dataset.

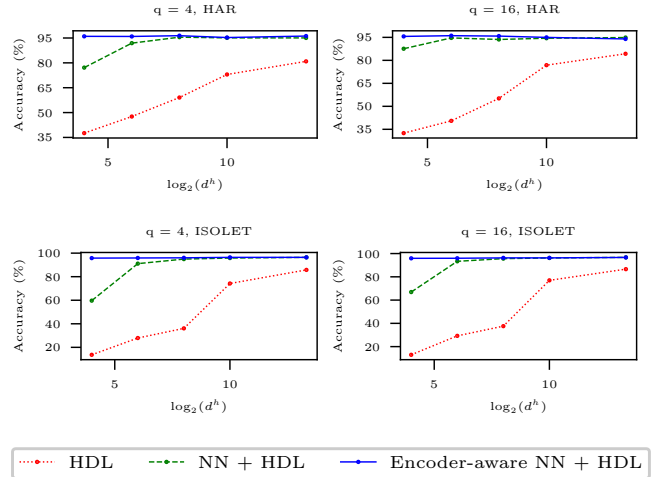| Machine Learning Model	Accuracy (Ratio of the Initial Training Data)			
HDL	85.76% (0.25)	85.76% (0.5)	85.76% (0.75)	85.76% (1)
SynergicLearning	86.21% (0.25)	91.21% (0.5)	94.03% (0.75)	95.77% (1)

Figure 6: Classification accuracy of different models on HAR and ISOLET datasets for different values of d^h and q.

and relatively low. For the SynergicLearning model, on the other hand, the accuracy keeps increasing when more data is provided to the NN in the initial training phase because it allows the NN to find higher quality features. This encourages less frequent, off-line updates to the NN for increasing the accuracy of the model.

Table 4: Comparison between the hardware metrics of SynergicLearning ($d^h = 16$) with pure HD ($d^h = 10,240$) over the ISOLET dataset on Xilinx UltraScale+ VU9P FPGA. The improvements of our approach compared to other approaches are shown in parantheses.

Approach	Implementation	BRAMs-18K (%)	DSPs-48E (%)	FFs (%)	LUTs (%)	Latency (μs)	Power (W)
SynergicLearning	NN+HD	1.8	15.0	0.8	5.1	23.3	5.3
Pure HD [16]	Parallel	0 (N/A)	0 (N/A)	11.0 (**93%**)	15.0 (**66%**)	49.5 (**53%**)	8.5 (**38%**)
	Sequential	0 (N/A)	0 (N/A)	11.0 (**93%**)	9.0 (**43%**)	788.7 (**97%**)	7.7 (**31%**)
NN [24, 25]	Systolic Array	1.7 (-6%)	15.0 (0%)	0.7 (-14%)	3.6 (-42%)	835.9 (**97%**)	5.1 (-4%)

5.5 The Hardware Cost of NN & HD Processing Modules

Fig. 7 shows the LUT utilization and latency of HD processing modules for different values of d^h while limiting the number of adders in each stage of tree adders to 16. It is observed that the latency grows very rapidly when increasing d^h to values required for meeting accuracy requirements. Additionally, to reduce the resource utilization for large values of d^h, we can change the architecture from a fully-parallel architecture to a vector-sequential architecture where all adders and counters operate in a sequential manner (compare Sequential Implementation with Parallel Implementation entries in Table. 4). While our parameterized architecture has a capability to generate both parallel and sequential-vector for HD processing module of SynergicLearning approach, we report the results for parallel implementation which delivers higher performance. Thanks to exteremly low d^h value in SynergicHD, the hardware overhead of parallel implementation is minimal.

Table 4 compares area utilization, latency, and power consumption of SynergicLearning at $d^h = 16$ with pure HD processing module at $d^h = 10,240$. SynergicLearning outperforms the fully-parallel pure HD processing module in terms of latency by a factor of 2.13x while yielding 1.60x lower power consumption. Compared to the vector-sequential implementation of HD processing module, SynergicLearning achieves 33.89x improvement in latency while yielding 1.45x lower power consumption.

It is worth mentioning that our designs are capable of achieving high clock rates (i.e. 344 MHz). The breakdown of different metrics between the NN and HD processing modules is as follows. The NN processing module consumes 93%, 100%, 87%, and 71% of the total consumed BRAMs-18K, DSPs-48E, FFs, and LUTs, respectively. The latency of the NN processing module is 23.12μs and the power consumption of the HD processing module is negligible compared to the NN processing module (i.e. less than 4% of total power consumption).
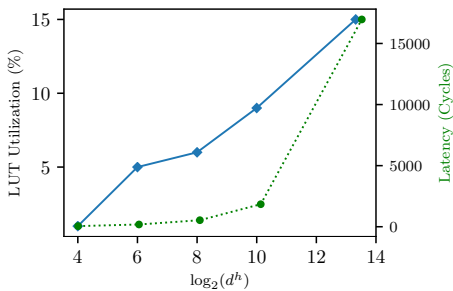

Figure 7: the LUT utilization and latency of HD processing modules for different values of d^h.

6 RELATED WORK

Kanerva [9] explains the advantages and mathematical properties of HD computing, and how data patterns should correspond in a systematic way to the entities they represent in the real world for achieving brain-like computing. Some of the prior work attempt to improve the performance of HD computing, either by increasing the obtained accuracy for some complex tasks, or enabling it to maintain the accuracy for lower dimensions. Authors in [10] propose a hierarchical HD computing framework, which enables HD to improve its performance using multiple encoders without increasing the cost of classification. In [11], authors utilize the mathematics of hyperdimensional spaces, and split each class hypervector into separate components and combine them into a reduced dimensional model. However, these works have not explored the effect of the feature extraction for low-dimensional input features.

Several studies in the literature explore hardware optimizations for implementing HD computing for different application domains. Authors in [26] propose a memory-centric architecture for the HD classifier with modular and scalable components, and demonstrate its performance on a language identification task. In [27], authors develop a programmable and scalable architecture for energy-efficient supervised classification using HD computing, and compare it with traditional architectures for a few conventional machine learning algorithms. The work in [28] explores architectural designs for the cleanup memory to facilitate energy-efficient, fast, and scalable search operation, and the proposed designs are evaluated for a language recognition application.

7 CONCLUSIONS

In this paper, we proposed SynergicLearning, in which by designing NNs that include some components of the HD models in their training loop, we trained high-quality feature extraction layers tailored to the HD learning model. By passing the input low-dimensional features through these layers before encoding them into the HD space, the number of dimensions of the HD space was reduced by two to three orders of magnitude, while maintaining the high classification accuracy, which led to less complex HD classifier. We also proposed and implemented an end-to-end fully-parametrized implementation of SynergicLearning for inference. Following the proposed hardware architecture, we achieved 2.13x improvement in terms of latency, while yielding 1.60x lower power consumption compared to pure HD computing.

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[2] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 2017.

[3] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, 2014.

[4] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Elsevier, 1989.

[5] Doyen Sahoo, Quang Pham, Jing Lu, and Steven CH Hoi. Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705*, 2017.

[6] Viktor Losing, Barbara Hammer, and Heiko Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 2018.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

[8] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International journal of computer vision*, 2015.

[9] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 2009.

[10] Mohsen Imani, Chenyu Huang, Deqian Kong, and Tajana Rosing. Hierarchical hyperdimensional computing for energy efficient classification. In *ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018.

[11] Justin Morris, Mohsen Imani, Samuel Bosch, Anthony Thomas, Helen Shu, and Tajana Rosing. CompHD: Efficient hyperdimensional computing using model compression. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019.

[12] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[13] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[14] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic DNN weight pruning framework using alternating direction method of multipliers. In *European Conference on Computer Vision (ECCV)*, 2018.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[16] Manuel Schmuck, Luca Benini, and Abbas Rahimi. Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2019.

[17] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

[18] Ron Cole, Yeshwant Muthusamy, and Mark Fanty. *The ISOLET spoken letter database*. Oregon Graduate Institute of Science and Technology, Department of Computer …, 1990.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

[20] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*. International Society for Optics and Photonics, 2019.

[21] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 2008.

[22] Andrey Ignatov. Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied Soft Computing*, 2018.

[23] Mohsen Imani, Sahand Salamat, Saransh Gupta, Jiani Huang, and Tajana Rosing. Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019.

[24] Gal Chechik, Uri Shalit, Varun Sharma, and Samy Bengio. An online algorithm for large scale image similarity learning. In *Advances in Neural Information Processing Systems*, 2009.

[25] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2017.

[26] Abbas Rahimi, Pentti Kanerva, and Jan M Rabaey. A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design (ISLPED)*, 2016.

[27] Sohum Datta, Ryan AG Antonio, Aldrin RS Ison, and Jan M Rabaey. A programmable hyper-dimensional processor architecture for human-centric IoT. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[28] Mohsen Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan M Rabaey. Exploring hyperdimensional associative memory. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017.