RefineHD: Accurate and Efficient Single-Pass Adaptive Learning Using Hyperdimensional Computing

Pere Vergés, Tony Givargis and Alexandru Nicolau
Department of Computer Science, University of California, Irvine
Irvine, California, United States of America
(pvergesb, givargis, nicolau)@uci.edu

***Abstract*—Hyperdimensional computing (HDC) is a computing framework that has gained significant attention due to its high efficiency and rapid training and inference of machine learning algorithms [1]. With its fast learning and inference capabilities, HDC shows excellent potential for IoT/Embedded systems. However, while HDC allows for fast single-pass learning [2], it suffers from weak classification accuracy, resulting from model saturation caused by excessive noise due to the addition of similar patterns. In this paper, we propose an adaptive learning method that surpasses accuracy and robustness compared to the state-of-the-art adaptive HDC model, while maintaining the same efficiency during both training and testing phases. Our method addresses the issue of saturation by selectively adding correctly classified samples only when their similarity to the existing patterns sufficiently differs from the class. Moreover, we achieve a robust model against noise and hardware failures, thanks to its HDC holographic properties. Through this approach, we achieve a remarkable average accuracy improvement of +2.8% across 126 datasets (with a maximum improvement of +26%). Furthermore, we observe a remarkable +6.6% improvement over the HDC baseline (with a maximum improvement of +67%), all while retaining the same inference efficiency.**

1. Introduction

Machine learning (ML) algorithms place significant demands on computational resources, including power, storage, and data. To address these requirements, computation is often offloaded to cloud-based data centers, which offer efficient processing of large amounts of data [3]. However, this demand arises from complex computations, such as gradient descent [4], that Neural Networks (NN) rely on. This approach presents challenges for Internet of Things (IoT) applications that operate on edge/embedded devices [5], [6]. These devices have limited resources and struggle to meet the high demands of traditional ML algorithms. First, performing real-time [7] online learning is limited by the inaccessibility of resources and the devices' dependence on unreliable battery sources [8]. Additionally, hardware failures and noise further exacerbate the challenges faced by these devices.

Hyperdimensional Computing (HDC) [9], also referred to as Vector Symbolic Architecture (VSA) [10], is a brain-inspired computational framework [11]. HDC is motivated by the immense number of neurons in the brain and operates by mapping input data into high-dimensional spaces, typically consisting of thousands (10,000) of dimensions [12]. It efficiently learns and identifies patterns within this high-dimensional representation [13]. One key advantage of HDC for IoT and embedded systems is its ability to address resource limitations and enhance hardware robustness [14], [15]. HDC is efficient due to its simplicity and potential for parallelization at both the thread and operation levels [16], [17]. These characteristics make HDC highly amenable to hardware optimization. Furthermore, due to the use of high dimensions in HDC, data representations are independently and identically distributed. Each dimension carries an equal amount of information, resulting in improved resilience to noise and hardware failures [18], making it particularly suitable for IoT systems.

HDC learning models offer the advantage of single-pass online training, where each data sample is processed only once. This learning approach is particularly well-suited for rapid real-time learning on data streams [19], [20], and applications that require low energy and high robustness are critical, such as EEG seizure detection, where HDC is the state-of-the-art [21]. However, in certain applications, HDC may not achieve high accuracy. To tackle this challenge, two approaches utilizing adaptive learning have been proposed [1], [2].

Adaptive learning overcomes the limitations of the naive single-pass learning approach by selectively incorporating samples into the model, thereby mitigating the impact of common data patterns that contribute to class hypervector saturation. In this paper, we introduce a novel adaptive learning model called RefineHD . This model improves upon the state-of-the-art HDC adaptive learning algorithm, OnlineHD [1], by reducing model variance and enhancing accuracy. Our proposed model employs a unique embedding technique using flocet embedding for key-value pair encoding [22], [23]. This embedding technique allows for a fixed number of neurons, which have a linear relation to a fixed number of active neurons, in contrast to the commonly used level hypervectors [24], resulting in an improved accuracy of +1.5% over 126 datasets. Furthermore, we compare our model to another popular encoding method that utilizes sinusoid projection [25] to map input features into hyperspace. This paper presents the following key contributions:

- Introduces RefineHD, a single-pass training framework that captures common patterns in each class hypervector while considering future changes in the associative memory. RefineHD updates the class hypervector for each input data by applying a weighted update based on the similarity to the incorrectly predicted class and the similarity to the second most similar class. It subtracts the weighted sample from the mispredicted class and adds it to the correct

class when the prediction is correct and the class similarity is lower than the average similarity of the incorrectly predicted classes. This mechanism ensures that RefineHD does not discard samples that would be misclassified after future updates on the associative memory. The proposed approach achieves a significant improvement since the state-of-the-art method, OnlineHD achieved a significant 2% over the baseline, and we achieve an additional 2.8% over OnlineHD across 126 datasets, with a maximum improvement of 26%.

- The flocet embedding for id-value encoding demonstrates superior accuracy compared to other commonly used encodings. The flocet encoding achieves an improvement of up to 1.5% compared to the level key-value pair encoding employed in other adaptive learning methods, on top of RefineHD.
- A comprehensive study that compares RefineHD to other proposed adaptive learning models, exploring the impact of different numbers of dimensions on capturing class patterns in an embedded device.
- An assessment of the robustness of the RefineHD model in comparison to other adaptive learning models.
- A study on how the accuracy of the model varies with different amounts of training data, comparing it to other existing adaptive learning models.

2. Related work

In the existing literature, two studies have focused on enhancing HDC classification through adaptive learning. It is worth noting that adaptive learning is a widely studied technique employed in various domains, including regenerative, compression, and multitask approaches. Therefore, improvements in adaptive learning techniques would not only benefit HDC but also enhance other approaches that utilize adaptive learning as a fundamental component of their algorithms. One notable study is AdaptHD [6], which employs adaptive learning by selectively incorporating samples only when the model incorrectly predicts the input data. This approach adds the sample to the correct class while subtracting it from the wrongly predicted class. Another relevant study is OnlineHD [5], which follows a similar principle but employs a weighted approach for adding and subtracting samples. The weight is determined based on the similarity between the class and the input sample.

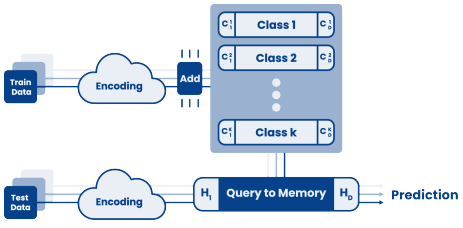

Figure 1: Hyperdimensional classification workflow.

3. Hyperdimensional Computing

Hyperdimensional Computing (HDC) originates from the quest to unravel the mechanisms underlying brain function and information representation in neurons [26]. This pursuit has spawned various models of geometric spaces known as hyperspaces, resulting in several distinct versions of the HDC framework, each defining its own unique hyperspace and symbolic information processing operations. Despite these differences, all HDC methods share a common goal: providing a resource-efficient alternative for computing tasks, especially suited for embedded systems in contrast to conventional machine learning techniques.

The hyperspace within HDC exhibits diverse variations, including binary, real, and complex spaces, each equipped with unique characteristics, operations, and similarity metrics, tailored to their respective domains [27]

In HDC, the fundamental building blocks for encoding data within the hyperspace are hypervectors. These hypervectors adhere to the holographic principle, with each dimension being independently distributed, ensuring robustness and uniform information distribution. Different types of basis-hypervectors are employed, including random-hypervectors selected quasi-orthogonally from the hyperspace [12], level hypervectors with linear correlation, and circular hypervectors with a circular or cyclical correlation [24].

Arithmetic operations in HDC rely on three highly parallelizable operations:

- *Binding* This operation involves element-wise multiplication of two hypervectors, resulting in a hypervector that differs from the originals. The operation can be represented as $\otimes : \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$.
- *Bundling* It entails the element-wise addition of two hypervectors, creating a hypervector similar to the originals. The operation can be denoted as $\oplus : \mathcal{H} + \mathcal{H} \rightarrow \mathcal{H}$.
- *Permuting* This operation [28] involves cyclically shifting the hypervector, with the shift being circular. In other words, the last element becomes the first if the shift is by one element. The operation is represented as $\Pi : \mathcal{H} \rightarrow \mathcal{H}$.

Data encoding in HDC involves the combination of multiple hypervectors to create intricate information representations. Diverse methods are employed for encoding various data types, such as text from characters [29], graphs from vertices and edges [30], time series from samples [31], and images from pixel values [32]. The subsequent section provides an overview of some of these established encoding strategies

4. Hyperdimensional Classification

HDC classification, like many other machine learning algorithms, can be divided into two main phases [33] [34]. The workflow of HDC classification is illustrated in Figure 1. The first phase is the training stage, where the model learns from the given input data. This phase may involve additional steps such as normalization, which is particularly relevant for the inference stage as it enhances computational efficiency. The second phase is the inference stage, where the trained model is used to predict new samples.

4.0.1. Training. The training process consists of three parts, as illustrated in Algorithm 1:

Encoding: The initial step of the training process involves encoding the input data samples into the hyperdimensional space. This can be done manually by utilizing the

Algorithm 1 HD Classification: Training

for each $i, label \in \mathcal{I}$ **do**
    $\mathcal{H} \leftarrow encode(i)$
    $M_{label} \leftarrow M_{label} + \mathcal{H}$
end for
$\mathcal{M} \leftarrow normalize(\mathcal{M})$

bundle, *bind*, and *permute* operations to create a customized mapping. Alternatively, commonly used approaches like the key-value pair encoding using level hypervectors or using sinusoid random projection can be employed, more in-depth explanation of the encoding process refer to this survey [34]. The purpose of encoding is to map the data into a high-dimensional space where each sample is quasi-orthogonal to one another. This ensures that if a similarity metric like cosine similarity is applied to two samples, the result will be close to zero, indicating no similarity between the two samples. It is important to note that this step is typically the most computationally intensive aspect of the training and testing stages.

Add to memory: The encoded data is then added to the associative memory, where there is a hypervector representation of each class. This stage is where adaptive learning can be applied to enhance the model's performance.

Algorithm 2 HD Classification: Inference

for each $i, label \in \mathcal{I}$ **do**
    $\mathcal{H} \leftarrow encode(i)$
    $pred \leftarrow argmax(\delta(\mathcal{H}, \mathcal{M}))$
end for

Normalize memory: After the completion of the training process, the associative memory is usually normalized to enable more efficient inference. Normalizing allows us to use the dot product similarity instead of the cosine similarity.

4.0.2. Inference. The inference process is divided into three steps as outlined in the pseudo-code of Algorithm 2. This process assumes that the associative memory has already been trained.

Encoding: Similar to the training process, the first step of the inference process involves encoding the input data to be mapped into the hyperdimensional space. It is crucial to use the same encoding for both the training and inference phases to ensure consistency and accuracy.

Query to memory: After encoding the input data, the next step in the inference process is to query the associative memory. This is done by computing the dot product between the encoded sample and the hypervectors representing each class in the associative memory. This computation generates a list of similarity measures between the sample and the different classes.

Prediction: Finally, from the list of similarity measures, the class with the highest similarity score is selected as the predicted class. If the predicted class label matches the actual class label, the prediction is considered correct. The process of retraining the model can be performed iteratively to enhance the accuracy of the model. However, this iterative process can be computationally expensive and may involve stopping criteria, such as when the accuracy of the last iterations does not vary or when the maximum number of iterations is reached.

Our work aims to propose an improvement over the existing single-pass learning algorithms to enhance the accuracy of single-pass learning. We intend to make our method more accurate than other proposed methods in the literature.

5. RefineHD Adaptive Learning

The baseline model for HDC classification, as described in Section 4, adopts a single-pass approach that is efficient and suitable for resource-constrained devices. However, this is a naive approach since it adds all instances to their respective class hypervectors, which can lead to model saturation (where we define saturation as reaching maximum information capacity) , due to the repeated addition of similar information of common patterns. When multiple instances of the same pattern are added to a class, it introduces noise and makes it challenging to accurately represent uncommon patterns. Consequently, the model may misclassify patterns that are not commonly found in the classes. To tackle this issue, previous works such as AdaptHD and OnlineHD have made advancements by excluding patterns of correctly classified instances during training. However, excluding some correctly classified patterns can impact the model's accuracy as the associative memory evolves with new patterns. Thus, there is a trade-off between maintaining accuracy and avoiding model saturation when incorporating patterns into the class hypervectors. Our proposed method takes a different approach by selectively adding correctly classified samples when their similarity deviates from their respective class. This method also improves the robustness to noise and hardware failure, such as bit flips, compared to the other methods. Embedded/IoT devices are often expected to operate in hostile environments, having higher noise and hardware errors. This issue is exaggerated when the computations are ML tasks [14], [15], [1]. However, thanks to HDC's high dimensionality and holographic representation such difficulties are addressed. Before jumping more into our method, we also propose a new embedding.

5.1. Flocet embedding

Mapping scalar representations to vectors has been employed in various domains, including stochastic computing, where boolean operations are applied to bit vectors to perform operations on scalars [35]. HDC operates in a similar manner by applying boolean operations to vectors. In this context, the flocet embedding technique has been proposed as a method for encoding sparse binary distributed data [36]. One significant advantage of this encoding is its systematic representation, which enables the creation of vectors on the fly. The activation of neurons in the flocet encoding can be expressed using the following function:

$$\mathcal{X}_i(v) = \mathcal{I}(i \in [v\mathcal{N}/\mathcal{Q}, v\mathcal{N}/\mathcal{Q} + w]), i = 1, N$$

Where v, represents a scalar, $\mathcal{I}(i, j)$ is the indicator function which takes 1 if $i < j$, otherwise is 0. $\mathcal{Q}$ is the number of gradations (rows) in the encoding. $\mathcal{N}$ is the number of dimensions (columns) of the encoding. And $w = \mathcal{N} - \mathcal{Q}$, represents the flocet width. Lastly, $\mathcal{B}$ represents the flocet length of the encoding. For HDC, it is important to set the

flocet length as $\mathcal{B} = \mathcal{N}/\in$. This will generate an embedding that preserves distance while maintaining a mean of 0 and variance of 1 among all the hypervector sets and having a fixed size, which will be equal to $\mathcal{N} * ((\mathcal{N}/\in) + \infty)$ as we show in Figure 2, for a flocet encoding of 6 dimensions. Finally, we show in Table 1 how the flocet encoding helps to better capture and generalize the input data information.

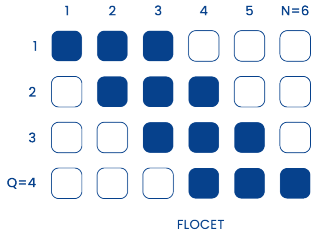

Figure 2: Flocet encoding using six dimensions.

5.2. RefineHD Single-Pass Learning

RefineHD is an innovative training approach designed to enhance single-pass online learning in HDC. The fundamental concept of RefineHD is to analyze each input sample and distinguish between common and uncommon patterns within its corresponding class, taking into account potential changes in the associative memory over time. By doing so, this method overcomes issues related to saturation and noise in the class hypervectors, while efficiently incorporating new pattern information without incurring additional time complexity.

In order to attain this objective, the RefineHD employs a sophisticated weighted update strategy. The primary goal of this strategy is to mitigate model saturation. Specifically, when the similarity between a sample and the correct class is high, we introduce a minor adjustment to the sample. However, when the sample deviates significantly from the correct class, we introduce a substantial adjustment to ensure the incorporation of that new pattern into the class's learning process. In the case of misclassification, a weighted portion of the input pattern is added to the correct class, while a weighted portion is subtracted from the misclassified class. Previous adaptive learning methods employed weighted addition based on the similarity between the input sample and its class, as well as weighted subtraction based on the similarity between the input sample and the misclassified class. In contrast, our approach aims to consider the similarity between the top two classes, promoting better generalization and increasing the orthogonality between classes. Consequently, when a misprediction occurs, our method adds the sample to the class using the following formula:

$$\mathcal{M}_{label} \leftarrow \mathcal{M}_{label} + (1 - \delta_{\mathcal{H}label})(1 - \delta_{\mathcal{H}label} - \delta_{\mathcal{H}pred})\mathcal{H}$$

$$\mathcal{M}_{pred} \leftarrow \mathcal{M}_{pred} - (1 - \delta_{\mathcal{H}pred})(1 - \delta_{\mathcal{H}label} - \delta_{\mathcal{H}pred})\mathcal{H}$$

This process helps reduce noise and maintains the integrity of class representations. Moreover, when an input pattern is correctly classified, it is only added to the class if its similarity to the class hypervector falls below a certain threshold. This threshold is determined by the average similarity of the wrongly predicted samples, which allows us to know the confidence level needed to accept a sample. Additionally, the weight of the addition to the class is determined by the distance between the top two classes, which can be calculated using the following equation:

$$\mathcal{M}_{label} \leftarrow \mathcal{M}_{label}(1 - \delta_{top_1} - \delta_{top_2})\mathcal{H}$$

The use of the top two classes helps to determine how confident is the prediction compared to the other classes, this way we can determine how much information should be added to the class.

The threshold serves as an indicator that even though the current classification is correct, future updates to the memory could potentially result in misclassifications. This adaptive learning approach aims to emulate the effect of retraining by assigning greater importance to uncommon patterns in the final class representation. However, unlike retraining, adaptive learning allows for efficient single-pass online learning without the requirement of iterative processes. By incorporating these adaptive learning techniques, the model can continually adapt to new patterns and improve its accuracy over time.

The steps involved in adding patterns to the associative memory are as follows: First, the input data is encoded into a hypervector $\mathcal{H}$. Then, the cosine similarity δ is computed by probing $\mathcal{H}$ against the associative memory $\mathcal{M}$ using the dot product formula:

$$\delta(\mathcal{H}, \mathcal{M}) = \frac{\mathcal{H} \cdot \mathcal{M}}{\|\mathcal{H}\| \cdot \|\mathcal{M}\|}$$

The cosine similarity represents the similarity in distance between the input hypervector and each class hypervector. It ranges from 0 to 1, with 1 indicating identical patterns and 0 indicating complete dissimilarity. Based on these similarity values, if the predicted output matches the class label of the input, the sample is included in the class only if the similarity value is lower than the average similarity of the mispredicted samples encountered thus far. This ensures that the model takes into account potential changes in the associative memory, recognizing that although the current prediction is accurate, the pattern may not be prevalent enough to maintain its characteristics as the training process progresses. The overall process is outlined in Algorithm 3.

Algorithm 3 RefineHD Single-Pass Learning: Training

```
for each i, label ∈ I do
    H ← encode(i)
    pred ← argmax(δ(H, M))
    if pred ≠ label then
        w ← (1 − δ_top1 − δ_top2)
        M_label ← M_label + (1 − δ_Hlabel) * w * H
        M_pred ← M_pred − (1 − δ_Hpred) * w * H
        update(δ_error)
    else
        if δ_pred ≤ δ_error then
            M_label ← M_label * w * H
        end if
    end if
end for
M ← normalize(M)
```

5.3. RefineHD Iterative Learning

In addition to the proposed single-pass online learning method, we have also implemented an iterative learning

version of the model. Iterative learning has the potential to enhance the accuracy of the model by allowing for multiple rounds of training and adjustment. However, it should be noted that iterative learning comes at the cost of increased time and power consumption. Therefore, it is suitable when online learning is not required and the embedded system has sufficient resources to handle the retraining process.

To adapt the method for iterative learning, it is necessary to reset the value of δ_{error} for each iteration. This ensures that the model can learn and adapt independently in each iteration without carrying over information from previous iterations. By resetting δ_{error}, we ensure that the model adapts and learns from scratch in each iteration.

The model will iterate until the number of iterations is reached or when the training accuracy has not changed significantly in the previous iterations, meaning that the memory has reached a stable value.

6. Experiments

In this section, we present the experiments conducted to evaluate the effectiveness of our method compared to other adaptive learning approaches, with a specific focus on its implications for ML in Embedded and IoT devices. These experiments range from measuring accuracy, generalization, and robustness to noise, hardware failures, and the amount of available training data that affect the adaptive learning methods. We conducted these experiments using a comprehensive set of popular datasets, totaling more than 120 datasets, and a Raspberry Pi.

6.1. Experimental Setup

The experiments were conducted using Torchhd, an HDC library built on top of PyTorch. The experiments were executed on two different platforms: firstly, on a Raspberry Pi B with four cores, and secondly, on a machine with 20 Intel Xeon Silver 4114 CPUs and 93 GB of RAM, which was used for evaluating the iterative versions due to their time consumption. To ensure consistency and reliability of the results, all experiments were repeated five times. The experiments were performed on a large number of datasets, including **121 datasets** from the UCI Repository, where we find different types of applications, from time series data to tabular data and different number of classes to classify, for the first set. The second set consisted of five datasets commonly used in the HDC classification literature. For the iterative versions, a maximum of 30 iterations was evaluated. The dimensionality used for all experiments was set to 10,000 dimensions. This decision comes from the field's standard used dimensions [26]. However, there are several applications that could be executed using fewer dimensions.

- **CIFAR10** [37]: CIFAR-10 is a widely-used dataset in image classification, containing 60,000 32x32 color images belonging to 10 classes.
- **EMGHandGestures** [38]: The EMG dataset is a collection of hand gestures performed by 36 subjects.
- **ISOLET** [39]: The ISOLET dataset is a collection of spoken letters of the alphabet. The dataset includes 26 classes, one for each letter of the alphabet.
- **MNIST** [40]: The MNIST dataset is a widely-used dataset in image recognition, containing a set of grayscale images of size 28x28 of handwritten digits from zero to nine.
- **PAMAP** [41]: The PAMAP2 Physical Activity Monitoring dataset is a collection of physical activity data captured by inertial measurement units and a heart rate monitor.

Using these two benchmarks will show the performance in a large number of different tasks ranging from small datasets to large datasets containing up to 4M instances.

6.2. Embedding accuracy comparison

This experiment aims to show how the embedding choice will affect the accuracy of the methods. To evaluate it, we will compare our proposed embedding (flocet) against the embedding used in the other adaptive methods (level) and another popular embedding (sinusoid). For the first two, we will use the key-value pair encoding, while the latter is itself an encoding. This evaluation is done using the 126 datasets. As we see in Table 1, the flocet encoding contributes with a 0.8-1.5% accuracy improvement on top of the RefineHD adaptive method compared to the encoding used in previous works.

Datasets	Level hashmap	Sinusoid	Flocet hashmap
HDC Benchmark	0.832 $\pm$ 0.033	0.839 $\pm$ 0.033	**0.847 $\pm$ 0.028**
UCI Benchmark	0.737 $\pm$ 0.063	0.736 $\pm$ 0.07	**0.742 $\pm$ 0.06**

TABLE 1: Embeddings accuracy comparison.

6.3. Methods accuracy and variance

In this experiment, we compare the performance of baseline and adaptive single-pass methods, as well as adaptive iterative methods found in the literature, against our RefineHD method. For a fair comparison, we apply the flocet encoding to all the methods. This approach will yield benefits for the other adaptive methods, as demonstrated in our previous experiment.

Table 2 provides a comparison of the single-pass methods and iterative methods, with a maximum of 30 iterations for the iterative methods, where the mean and variance have been normalized over all the datasets. The results demonstrate that the average accuracy of the RefineHD single-pass method exhibits a significant improvement of 1.8% on the HDC Benchmark, surpassing the current state-of-the-art adaptive method, and a 6.5% improvement compared to the baseline. On the UCI Benchmark, which consists of 121 datasets, the achieves a 2.8% accuracy improvement compared to the state-of-the-art method and a 4.9% improvement compared to the baseline.

Regarding iterative training, the improvements are marginal, as the iterative process allows other methods to generalize better over multiple iterations. However, it is important to note that the accuracy of iterative models on the HDC Benchmark remains lower than the using single-pass learning. On the UCI Benchmark, the iterative process yields an approximately 2% improvement compared to the RefineHD single-pass approach. Nevertheless, this improvement comes at the cost of significantly increased training time, proportional to the number of iterations, making iterative training unsuitable for online learning on resource-constrained devices.

Method	HDC (5 datasets)	UCI (121 datasets)
Baseline	0.783 ± 0.0026	0.699 ± 0.0068
Baseline (Pi)	0.785 ± 0.0056	0.699 ± 0.0001
AdaptHD	0.821± 0.01	0.713 ± 0.057
AdaptHD (Pi)	0.819 ± 0.0101	0.715 ± 0.007
OnlineHD	0.831 ± 0.0078	0.720 ± 0.058
OnlineHD (Pi)	0.825 ± 0.0116	0.722 ± 0.005
RefineHD	**0.849 ± 0.006**	**0.748 ± 0.039**
RefineHD (Pi)	**0.847 ± 0.0056**	**0.743 ± 0.0029**
AdaptHD Iterative	0.836 ± 0.004	0.758 ± 0.023
OnlineHD Iterative	0.847 ± 0.002	0.76 ± 0.03
RefineHD Iterative	**0.851 ± 0.0027**	**0.767 ± 0.018**

TABLE 2: Benchmarks average accuracy (10000 dimensions).

Method	CIFAR	EMG	ISOLET	MNIST	PAMAP
Baseline	0.29	0.981	0.887	0.832	0.927
AdaptHD	0.333	0.995	0.86	0.923	0.997
OnlineHD	0.345	0.997	0.89	0.926	0.998
RefineHD	**0.407**	**1**	**0.9**	**0.938**	**0.999**
AdaptHD Iterative	0.332	0.997	0.928	0.934	0.992
OnlineHD Iterative	0.349	0.997	0.938	**0.957**	**0.999**
RefineHD Iterative	**0.419**	**0.999**	**0.94**	0.91	0.989

TABLE 3: Accuracy on the HDC benchmark datasets.

Table 3 provides a detailed accuracy comparison for each dataset in the HDC popular applications, considering both single-pass and iterative adaptive learning methods. The table demonstrates that among the single-pass adaptive learning methods, RefineHD consistently achieves superior accuracy across all datasets. It achieves a maximum improvement of 6.2% over the state-of-the-art method and an impressive 11.7% improvement over the baseline method.

Additionally, for the UCI benchmark, a maximum improvement of 26% over OnlineHD and 67% over the baseline. However, in the case of the MNIST and PAMAP datasets, OnlineHD performs better than RefineHD. This highlights the importance of considering the specific characteristics of each dataset when selecting the most appropriate learning method.

6.4. Methods performance

The execution time, encompassing both training and testing phases, primarily arises from the encoding and comparison operations between input samples and hypervector classes. Given that all methods share these fundamental tasks, disparities in execution time predominantly stem from variations in the number of comparisons, additions, and subtractions performed by each method. Importantly, these supplementary computational operations exert only a marginal influence on the overall execution time, resulting in comparable performance efficiencies among HDC adaptive learning methods. In terms of memory usage, we see the same outcome; all methods have very similar memory consumption.

6.5. Varying dimensions

To explore the generalization capabilities of the adaptive methods, we conducted a study by varying the number of hyperdimensional dimensions for each application. Figure 3 showcases that our RefineHD method consistently achieves higher accuracy across all dimensions compared to other adaptive learning approaches. This finding demonstrates the effectiveness of our technique in enabling faster and improved generalization of the different classes, regardless of the dimensionality.

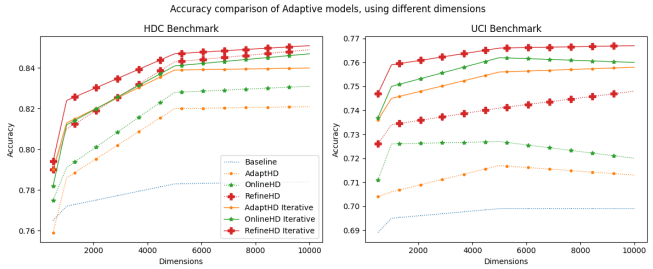

Figure 3: Evaluation over different dimensions on adaptive learning.

6.6. Partial data experiment

In this experiment, we aimed to demonstrate how our proposed RefineHD model effectively captures data patterns. We achieved this by evaluating the model's performance under varying amounts of training data. The results, as depicted in Figure 4, clearly show that the RefineHD model consistently captures more significant patterns compared to other adaptive learning methods across all different amounts of training data. This happens for both the single-pass and iterative versions.

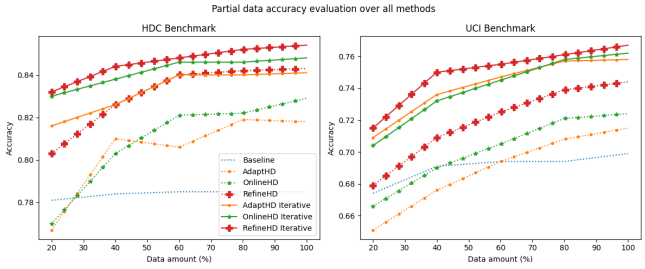

Figure 4: Partial data accuracy evaluation over adaptive learning methods.

6.7. Robustness to hardware failure (bit flips)

When using binary vectors, an error consists of a bit flip, which can completely change what a vector represents. However, by leveraging the high-dimensionality and holographic representation, we aimed to demonstrate that RefineHD exhibits high robustness to errors and bit flips. This robustness is particularly crucial for Embedded/IoT devices, which often face such issues [14], [15], [1], especially when performing ML tasks, where the algorithms' noise robustness is typically low [42]. The final experiment investigates how the proposed model responds to hardware failures caused by bit flips (failed dimensions) and compares it to other adaptive learning models proposed in the literature. In this experiment, we have flipped a percentage of bits selected at random. Figure 5 presents the results, and we observe that the baseline takes more time to deteriorate in performance than the other methods. This is expected since the accuracy of this method is lower, making it harder to degrade its performance. Additionally, we see that most methods behave similarly as the error rate increases. However, our methods consistently exhibit higher accuracy across all different amounts of bit failures, showcasing their robustness to such errors.

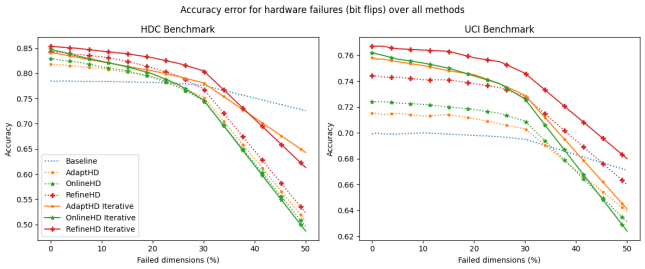

Figure 5: Robustness evaluation over adaptive learning methods.

7. Conclusions

This work proposes an adaptive online learning method that improves accuracy and robustness compared to the state-of-the-art. The model can perform more accurate predictions while being a single-pass lightweight solution for resource-constrained devices. The RefineHD method employs single-pass learning, selectively adding uncommon patterns and removing common patterns to reduce model saturation while considering future changes in the model. This is achieved by adding weighted samples to the memory based on the similarity to the classes and the top 2 predictions. Additionally, correctly predicted classes are added only when the similarity to the class is lower than the mean of the similarity of a sample with its class when the samples are misclassified. The RefineHD method achieves an additional 2.8% improvement over OnlineHD across 126 datasets, with a maximum improvement of +26%, all while maintaining the same time complexity. Moreover, this work proposes using flocet embedding combined with key-value pair encoding to achieve better accuracy results on the 126 datasets tested, providing 1.5% higher accuracy than the key-value pair encoding using level embedding used in previous adaptive learning works. These findings highlight the effectiveness of the RefineHD method in improving accuracy and robustness while being well-suited for real-time, single-pass learning in resource-constrained environments like Embedded and IoT systems. In our forthcoming research, we are committed to conducting a comprehensive assessment of HDC's robustness. Specifically, we intend to delve deeper into the impact of various dimensions on the model's degradation. Furthermore, we plan to design and evaluate more extensive datasets to explore the capabilities of HDC. Additionally, our research agenda encompasses the exploration of methods to enhance the interpretability of HDC.

References

[1] H.-C. et al., "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 (DATE)*, 2021, pp. 56–61.

[2] M. e. a. Imani, "Adapthd: Adaptive efficient training for brain-inspired hyperdimensional computing," in *2019 (BioCAS)*, 2019.

[3] A. Adadi, "A survey on data-efficient algorithms in big data era," *Journal of Big Data*, vol. 8, pp. 1–54, 2021.

[4] N. P. Nieves and D. F. M. Goodman, "Sparse spiking gradient descent," *CoRR*, 2021. [Online]. Available: https://arxiv.org/abs/2105.08810

[5] A. e. a. Azari, "Self-organized low-power iot networks: A distributed learning approach," in *2018 IEEE GLOBECOM*, 2018.

[6] F. e. a. Samie, "Computation offloading and resource allocation for low-power iot edge devices," in *2016 IEEE 3rd WF-IoT*.

[7] B. et al, "Online learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing," *IEEE Transactions on Biomedical Circuits and Systems*, 2019.

[8] A. M. et al., "Odd-ecc: on-demand dram error correcting codes," *Proceedings of the International Symposium on Memory Systems*, 2017.

[9] 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.

[10] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," in *(ICCS/ASCS)*, 2003.

[11] H. Jaeger, "Towards a generalized theory comprising digital, neuromorphic and unconventional computing," *Neuromorphic Computing and Engineering*, 2021.

[12] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[13] D. e. a. Kleyko, "Vector symbolic architectures as a computing framework for nanoscale hardware," 2021.

[14] Y. e. a. Kim, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 DATE*, 2020.

[15] S. e. a. Salamat, "Accelerating hyperdimensional computing on fpgas by exploiting computational reuse," *IEEE Transactions on Computers*.

[16] J. e. a. Kang, "Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training," in *2022 27th ASP-DAC*, 2022.

[17] P. V. et al., "Hdcc: A hyperdimensional computing compiler for classification on embedded systems and high-performance computing," 2023.

[18] S. e. a. Zhang, "Assessing robustness of hyperdimensional computing against errors in associative memory : (invited paper)," in *(ASAP)*, 2021.

[19] A. e. a. Hernández-Cano, "Real-time and robust hyperdimensional classification," ser. GLSVLSI '21, 2021.

[20] U. e. a. Pale, "Exploration of hyperdimensional computing strategies for enhanced learning on epileptic seizure detection," in *2022 44th EMBC*.

[21] A. e. a. Burrello, "One-shot learning for ieeg seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *2018 (BioCAS)*.

[22] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *Circuits and Systems Magazine*, 2020.

[23] A. T. et al., "A theoretical perspective on hyperdimensional computing," *Journal of Artificial Intelligence Research*, oct 2021.

[24] I. e. a. Nunes, "An extension to basis-hypervectors for learning from circular data in hyperdimensional computing," 2022.

[25] Z. e. a. Zou, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *SC21*.

[26] P. Kanerva, "Computing with 10,000-bit words," in *Allerton Conference on Communication, Control, and Computing*. IEEE, 2014, pp. 304–310.

[27] K. Schlegel, P. Neubert, and P. Protzel, "A comparison of vector symbolic architectures," *Artificial Intelligence Review*, pp. 1–33, 2021.

[28] P. Vergés, I. Nunes, M. Heddes, T. Givargis, and A. Nicolau, "Accelerating the permute and n-gram operations for hyperdimensional learning in embedded systems," EasyChair Preprint no. 11190, EasyChair, 2023.

[29] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2016, pp. 64–69.

[30] I. e. a. Nunes, "Graphhd: Efficient graph classification using hyperdimensional computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.

[31] A. e. a. Rahimi, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[32] M. et al., "Performance analysis of hyperdimensional computing for character recognition," in *International Symposium on Multimedia and Communication Technology (ISMAC)*. IEEE, 2019, pp. 1–5.

[33] D. K. et al, "Density encoding enables resource-efficient randomly connected neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, aug 2021.

[34] P. Vergés, M. Heddes, I. Nunes, T. Givargis, and A. Nicolau, "Classification using hyperdimensional computing: A review with comparative analysis," *Computing in Science & Engineering*.

[35] A. e. a. Alaghi, "Computing with randomness," *IEEE Spectrum*, 2018.

[36] D. e. a. Rachkovskij, "Sparse binary distributed encoding of scalars," *Journal of Automation and Information Sciences*, 2005.

[37] A. Krizhevsky, "Learning multiple layers of features from tiny images."

[38] A. e. a. Rahimi, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *2016 IEEE ICRC*, pp. 1–8.

[39] M. e. a. Imani, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE ICRC*.

[40] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," 2012.

[41] A. e. a. Reiss, "Introducing a new benchmarked dataset for activity monitoring," ser. ISWC '12, 2012.

[42] S. H. et al., "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016.