Dynamic Hyperdimensional Computing for Improving Accuracy-Energy Efficiency Trade-offs

Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
{frankchuang, kevin}@access.ee.ntu.edu.tw, andywu@ntu.edu.tw

Abstract—Brain-inspired Hyperdimensional (HD) computing is an emerging technique that computes with either binary or integer HD vectors. However, both vector representations confront an extreme trade-off between accuracy and energy efficiency. This issue limits the generalizability of HD computing for many applications. In this paper, we propose a threshold-based dynamic HD computing framework (TD-HDC) to improve the accuracy-energy efficiency trade-offs. Standard HD computing always executes the same processing flow regardless of input data. On the contrary, TD-HDC dynamically selects the execution path between the binary and integer HD models based on the classification difficulty of input data. In other words, TD-HDC utilizes both HD models and manages to efficiently allocate their computational resources. On the MNIST dataset, we demonstrate that our proposed framework is flexible and can reduce energy consumption and execution time by 51.3% and 15%, respectively, under the same accuracy level.

Index Terms—Brain-inspired computing, hyperdimensional computing, dynamic model, energy efficiency

I. INTRODUCTION

In the era of the Internet of Things (IoT), the massive streams of data are produced daily by sensor nodes. Machine learning on edge provides a feasible way to leverage these vast amounts of data. However, existing learning algorithms usually require considerable computational resources and memory storage, which makes them impractical to be deployed on resource-constrained edge devices.

Brain-inspired Hyperdimensional (HD) computing [1] is a novel technique and shows great potential as a lightweight classifier in the field of IoT. HD computing emulates patterns of neural activities by computing with high-dimensional (e.g. thousands) HD vectors. Through exploiting the mathematical properties of vectors in a high-dimensional space, HD computing exhibits many remarkable characteristics, including high energy efficiency, fast learning ability, and robustness to hardware failures [2]-[7]. Moreover, HD computing has shown successful progress in various real-world applications, such as emotion recognition [8], seizure detection [9], DNA sequencing [10], speech recognition [11], and human activity recognition [12]. Recently, [13] enables HD computing to be executed under an ultra-low-power condition, and they fabricate the HD model on emerging 3D nanoscale devices to achieve higher energy efficiency. These properties and advantages both make HD computing suitable for efficient deployment on edge devices.

In Fig. 1, we illustrate the processing flow of training and inference in general HD computing, which involves the transformation of data into HD vectors. Each component of HD vectors can be represented as a binary number for energy efficiency. Recently, several works argue that HD computing requires the usage of integer HD vectors to achieve more favorable performance [14]-[16]. However, compared with binary vectors, working with integer values greatly increases the energy costs of the HD model. In other words, no matter which data representation for HD vectors is employed, existing HD computing algorithms inevitably confront an extreme trade-off between model performance and energy efficiency. This issue limits the generalizability of HD computing for many applications. Hence, an effective framework for improving the trade-off between accuracy and energy efficiency is necessary for HD computing.

In this paper, we propose a threshold-based dynamic HD computing framework (TD-HDC) to effectively improve the accuracy-energy efficiency trade-off. Different from standard HD computing which always executes the same processing flow regardless of input data, we enable HD computing to dynamically allocate computational resources of both binary and integer HD models based on input data. The main contributions of this paper are summarized as follows:

1) Selecting the execution path between the binary and integer HD model based on input data: We propose TD-HDC as a dynamic HD computing framework that can select the execution path between the binary and integer HD models. In contrast to static HD computing which always executes the same processing flow, dynamic HD computing utilizes both HD models and efficiently allocate...
their computational resources based on the classification difficulty of input data. On the MNIST dataset, we demonstrate that the proposed framework is flexible and can reduce 51.3% of energy consumption and 15% of execution time under the same accuracy level.

2) Development of the two-stage retraining mechanism: To further improve the overall energy efficiency, we propose a two-stage retraining mechanism for the training stage of TD-HDC. With this mechanism, TD-HDC can further reduce 6.22% of energy consumption compared with that without the mechanism.

The rest of this paper is organized as follows. We briefly review the processing flow of HD computing in Section II. Our proposed TD-HDC and the two-stage retraining mechanism are presented in Section III. Section IV shows our simulation results and discussion. Finally, we conclude this paper in Section V.

II. HYPERDIMENSIONAL COMPUTING

A. Mapping to HD Space

In HD computing, the goal of mapping is to project the information of a feature vector \( F = [f_1, f_2, \ldots, f_m] \) to D-dimensional HD vectors. Each entry of \( F \) is a feature component corresponding to a pair of unique feature ID and a feature value, and they are mapped to HD vectors through the Item Memory (IM) and the Continuous Item Memory (CiM), respectively. \( IM = \{ID_1, ID_2, \ldots, ID_m\} \) maps the feature ID of the \( i \)-th entry of \( F, f_i \), to a binary HD vector \( ID_i \in \{0, 1\}^D \). Note that any two different HD vectors in IM are nearly orthogonal, i.e., \( \text{Hamming}(ID_j, ID_k) \approx 0.5, j \neq k \), where \( \text{Hamming}(\cdot, \cdot) \) is the normalized Hamming distance between two vectors. To map continuous feature values of each feature component into HD space, the CiM first quantizes the range of values into \( l \) levels. Each HD vector in \( \text{CiM} = \{L_1, L_2, \ldots, L_l\} \) should preserve the spatial relation of feature values. Namely, if feature values of \( f_i \) and \( f_j \) are close, their corresponding HD vectors \( S_i \in \{L_1, L_2, \ldots, L_l\} \) and \( S_j \in \{L_1, L_2, \ldots, L_l\} \) have a relatively small Hamming distance. To achieve this, two randomly generated HD vectors \( L_i, L_i \in \{0, 1\}^D \) are first assigned to the maximum and minimum value of each feature component, respectively. After that, \( D/l \) randomly selected bits are flipped to generate an HD vector for the next level, and so on. The bit-flipping approach ensures a high correlation between adjacent levels, while \( L_i \) is orthogonal to \( L_i \). Through the IM and the CiM, the \( i \)-th entry of \( F, f_i \), would be mapped to a pair \((ID_i, S_i)\).

B. Encoding

A set of two-vector pairs \( l = \{ID_1, S_1, \ldots, ID_m, S_m\} \) is generated after mapping each feature component of \( F \) to HD vectors. HD computing utilizes well-defined bitwise vector operations to encode the pairs in \( l \) to a single representative HD vector \( T \in \{0, 1\}^p \), which is expressed as

\[
T = \left[ ID_1 \oplus S_1 + ID_2 \oplus S_2 + \cdots + ID_m \oplus S_m \right],
\]

where \( \oplus \) is defined as the bitwise XOR for binding two HD vectors, + is defined as bitwise addition for representing sets, and \( [\cdot] \) is defined as the majority function to binarize each component of HD vectors with the threshold \( m/2 \).

C. Training

In the training phase, all encoded HD vectors of the training examples belonging to the same class are summed up to form the representative class vector. For the \( p \)-th class, the corresponding class vector \( C_p \) is computed as

\[
C_p = \sum_{q=1}^{n_p} T^q_p = T^1_p + T^2_p + \cdots + T^{n_p}_p,
\]

where \( n_p \) is the number of training samples of the \( p \)-th class and \( T^q_p \) is the \( q \)-th encoded HD vector belonging to the \( p \)-th class \((q = 1, 2, \ldots, n_p)\). For a \( k \)-class classification task, \( k \) class vectors denoted as \( \{C_1, C_2, \ldots, C_k\} \) are generated and stored in the associative memory (AM). In HD computing, each component of a class vector can be an integer or a binary number. In this paper, the AM storing integer class vectors computed by (2) is called integer HD model whereas integer class vectors are denoted as \( \{C_1, C_2, \ldots, C_k\} \). The AM binarized from the integer HD model is called binary HD model whose binary class vectors are denoted as \( \{c_{1}^{\text{bin}}, c_{2}^{\text{bin}}, \ldots, c_{k}^{\text{bin}}\} \).
D. Inference

In the inference phase, a testing sample is first encoded as a query vector $Q \in \{0,1\}^D$ by the same processing flow of mapping and encoding in the training phase. Next, HD computing evaluates the similarity between $Q$ and all class vectors stored in AM. The class with the highest similarity is selected as the prediction. For the integer HD model, the similarity metric is cosine similarity. For the binary HD model, the similarity metric is simplified as the Hamming distance, which can be simply implemented by XOR gates and counters. Generally speaking, the integer HD model achieves more favorable performance at the expense of huge energy cost for computing cosine similarity. The binary HD model has relatively higher energy efficiency but significantly degrades accuracy due to information loss from binarization [14]-[16].

III. PROPOSED THRESHOLD-BASED DYNAMIC HYPERDIMENSIONAL COMPUTING

A. Dynamic HD Computing Framework

In this work, we utilize both binary and integer HD models to establish a dynamic framework for HD computing to improve the accuracy-energy efficiency trade-off. The core idea of TD-HDC is to dynamically select the execution path between the binary and integer HD models based on the classification difficulty of samples. In other words, we employ the binary HD model to classify those “easy” samples so that a considerable amount of unnecessary computation can be avoided. For those “difficult” samples, the integer HD model is utilized to preserve decent performance. Therefore, different from standard HD computing which always executes the same processing flow, TD-HDC can effectively allocate the computational resources of both HD models. To efficiently evaluate the difficulty of a sample, we define the classification confidence value $\tau$ of the binary HD model as

$$\tau = \arg\min_p \text{Hamming} (Q, C^{\text{bin}}_p) - \text{Hamming} (Q, C^{\text{bin}}_{p^*}),$$

where

$$p^* \equiv \arg\min_p \text{Hamming} (Q, C^{\text{bin}}_p).$$

$\tau$ is used as an evaluation of the difficulty of correctly classifying a sample. The larger $\tau$ is, the more confident we are to guarantee that the binary HD model can make the right prediction. Fig. 3 shows the histogram of classification confidence values of the binary HD model evaluated on the MNIST handwritten digit dataset. The red and blue bars correspond to the samples that are wrongly and correctly classified by the binary HD model, respectively.

![Fig. 3. A histogram of classification confidence values of the binary HD model evaluated on the MNIST dataset. The red and blue bars correspond to the testing samples that are wrongly and correctly classified by the binary HD model, respectively.](image)

The overall energy efficiency of TD-HDC depends on the ratio of data being early classified by the binary HD model. Therefore, if the classification precision of the binary HD model is improved, more samples can be early classified to achieve higher energy efficiency. The previous retraining method [11] is proposed to enhance the performance of a binary HD model. Here, we briefly review how it works. The binary HD model starts retraining by iteratively validating on the training set. If the training sample is correctly classified, no change would be made. However, if the query vector $Q$ of the training sample is misclassified, then $Q$ is subtracted from the integer class vector corresponding to the incorrectly predicted class $C^{\text{int}}_{\text{wrong}}$ and added to the correct integer class vector $C^{\text{int}}_{\text{correct}}$, which can be expressed as

$$C^{\text{int}}_{\text{correct}} = C^{\text{int}}_{\text{correct}} + Q,$$

$$C^{\text{int}}_{\text{wrong}} = C^{\text{int}}_{\text{wrong}} - Q.$$

Note that although the similarity check is performed on the binary HD model, we only update the integer one. After several iterations across all training data, the new binary HD model is...
acquired by binarizing those updated integer class vectors.

In TD-HDC, the binary HD model is obtained by binarizing the integer HD model. Therefore, the integer class vectors stored in the integer HD model would be updated based on the classification results of the binary HD model. Although the retraining approach mentioned above can enhance the performance of the binary HD model; however, this approach would degrade the performance of the integer HD model, since the updating criteria are not based on the classification results of the integer HD model. To improve the performance of the binary HD model while preserving that of its integer counterpart, we propose a two-stage retraining mechanism for TD-HDC, as shown in Fig. 2(b). After initial training, TD-HDC obtains both the binary and integer HD models. In the first retraining stage, we adjust the class vectors in the integer HD model based on the classification results of the binary HD model (❶). In the second retraining stage, we retrain the integer HD model based on the classification results of itself to compensate for the performance degradation in the first stage (❷). The first and second retraining stages iterate over the training data until both HD models converge. The convergence condition is defined as that the accuracy of both HD models does not improve in the last five iterations. After convergence, the updated integer HD model is binarized to obtain the new binary HD model (❸). The proposed two-stage retraining mechanism is summarized in Algorithm 2.

**Algorithm 1** Inference of TD-HDC

**Input:** Testing data $N_{test}$, threshold $t$

**Output:** Prediction $O$

1: Transform $N_{test}$ into a query vector $Q$ by (1)
2: The binary HD model makes prediction $O_{bin}$ on $Q$
3: Calculate the classification confidence value $\tau$ of $Q$
4: if $\tau > t$ do
5:   **return** prediction $O = O_{bin}$
6: else do
7:   The integer HD model makes prediction $O_{int}$ on $Q$
8:   **return** prediction $O = O_{int}$

**Algorithm 2** Two-stage retraining mechanism of TD-HDC

**Input:** Training data $N_{train}$

**Output:** The updated binary and integer HD models

1: Transform $N_{train}$ into a query vector $Q$
2: while both HD models still improve in the last five iterations do
3:   The binary HD model classifies $N_{train}$
4: if the binary HD model predicts wrongly do
5:   $C_{int\ wrong} = C_{int\ wrong} - Q$
6:   $C_{int\ correct} = C_{int\ correct} + Q$
7: else do
8:   The integer HD model classifies $N_{train}$
9: if the integer HD model predicts wrongly do
10:  $C_{int\ wrong} = C_{int\ wrong} - Q$
11:  $C_{int\ correct} = C_{int\ correct} + Q$
12: **return** the updated binary and integer HD models

Fig. 4. Averaged accuracy and the ratio of data transferred to the integer HD model under different confidence thresholds ($t$).

Fig. 5. Normalized energy consumption and normalized execution time under different confidence thresholds ($t$).

IV. EXPERIMENT SETTINGS AND SIMULATION RESULTS

A. Dataset and Simulation Settings

We verify the effectiveness of TD-HDC on the MNIST handwritten digit database, a standard benchmark for machine learning models. The proposed framework for HD computing is implemented in Python running on AMD Core TR-2950X processor with 128 GB memory. We compare the classification accuracy, energy efficiency, and execution time of TD-HDC with the two-stage retraining mechanism to a pure integer HD model and a pure binary HD model. To measure the energy efficiency, we refer to [17] to estimate the ratio of energy consumption between the binary and integer HD models, and the result is approximately 0.032:1. We set the dimensionality of HD computing as $D = 5000$ and the quantization level as $l=10$, where the performance of both HD models saturates. All experiments are conducted over 100 independent runs to obtain the averaged simulation results.
B. Accuracy Analysis with Different Confidence Thresholds

Fig. 4 illustrates the averaged accuracy of TD-HDC and the ratio of data transferred to the integer HD model under different confidence thresholds. When the threshold is set as 0.1, all testing data are transferred to the integer HD model. This demonstrates that the averaged accuracy of the pure integer HD model is 98.09%. On the contrary, all testing data are classified by the binary HD model when the threshold is set as 0. Compared with the pure integer HD model, the averaged accuracy of the pure binary one significantly degrades by 9.2% due to information loss from binarization. In TD-HDC, we enable HD computing to dynamically select the execution path between the binary and integer HD models based on the classification confidence values. As shown in Fig. 4, increasing the confidence threshold would transfer more data with low classification confidence values to the integer HD model for more reliable classification. This accordingly enhances the averaged classification accuracy of TD-HDC. When the threshold is set as 0.03, although only 47% of data are transferred to and classified by the integer HD model, TD-HDC shows comparable accuracy of 97.8% to the pure integer HD model. This indicates that the other 53% of data are easy enough to be classified by the binary HD model.

C. Analysis of Energy Consumption and Execution Time

Fig. 5 shows the impact of the confidence threshold on the energy consumption and execution time of TD-HDC, which are both normalized to those of the pure integer HD model. When the threshold is set as a lower value, more testing data would be early classified by the binary HD model and lower energy consumption and execution time can be achieved. On the other hand, when the threshold is set as a higher value, TD-HDC consumes more energy and execution time due to the computation of cosine similarity. Table I shows the averaged energy consumption and execution time between different HD models. TD-HDC with the two-stage retraining mechanism further reduces 6.22% of energy consumption compared with that without the mechanism, since the binary HD model can achieve better classification precision. When t = 0.03, TD-HDC effectively allocates the computation resources of both HD models and reduces 51.3% of energy consumption and 15% of execution time with comparable accuracy to that of the pure integer HD model. When t = 0.02, TD-HDC reduces more energy consumption by 66.9% with a tolerable accuracy drop. TD-HDC can further achieve higher energy efficiency when t = 0.01. Based on the experimental results, we demonstrate that, by adjusting the threshold, TD-HDC is flexible to improve the trade-off between accuracy and energy efficiency.

V. CONCLUSION

In this paper, we propose a threshold-based dynamic HD computing (TD-HDC) to effectively improve the accuracy-energy efficiency trade-off of HD computing. TD-HDC dynamically selects the execution path between the binary and integer HD models based on the classification difficulty of input data. From the experimental results, we demonstrate that TD-HDC is flexible and can reduce energy consumption and execution time by 51.3% and 15%, respectively.

### TABLE I - SUMMARY OF DIFFERENT HD MODELS

<table>
<thead>
<tr>
<th>Model</th>
<th>Averaged Accuracy</th>
<th>Normalized Energy Consumption</th>
<th>Normalized Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure integer HD model</td>
<td>98.09%</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TD-HDC (t=0.03)</td>
<td>97.80%</td>
<td>0.4877</td>
<td>0.8510</td>
</tr>
<tr>
<td>TD-HDC (t=0.02)</td>
<td>97.42%</td>
<td>0.3316</td>
<td>0.8022</td>
</tr>
<tr>
<td>TD-HDC (t=0.01)</td>
<td>95.58%</td>
<td>0.1909</td>
<td>0.7598</td>
</tr>
<tr>
<td>Pure binary HD model</td>
<td>88.92%</td>
<td>0.0303</td>
<td>0.7051</td>
</tr>
<tr>
<td>Two-stage Retraining</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TD-HDC (without two-stage retraining, t=0.03)</td>
<td>97.74%</td>
<td>0.5499</td>
<td>0.8658</td>
</tr>
<tr>
<td>TD-HDC (with two-stage retraining, t=0.03)</td>
<td>97.80%</td>
<td>0.4877</td>
<td>0.8510</td>
</tr>
</tbody>
</table>

### REFERENCES