OTFGEncoder-HDC: Hardware-efficient Encoding Techniques for Hyperdimensional Computing

Mahboobe Sadeghipour Roodsari, Jonas Krautter, and Mehdi Tahoori
Institute of Computer Engineering, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{mahboobe.sadeghipourrudsari, jonas.krautter, mehdi.tahoori}@kit.edu

***Abstract*—Hyper-Dimensional Computing (HDC), a brain-inspired computing paradigm for cognitive tasks, is especially suited for resource-constrained edge devices due to its hardware-efficient and fault-resistant inference. However, existing HDC approaches require large amounts of memory, resulting in high power consumption, limiting their use in edge devices. We offer a hardware-aware encoding where computation parameters in hardware implementations can be reproduced on-the-fly through low-overhead cyclic digital circuits, significantly reducing memory utilization and subsequently power consumption.**

I. Introduction

The integration of machine learning in various domains, particularly at the edge, is crucial for IoT applications. While real-time processing, energy efficiency, and fault tolerance are essential in such applications, edge devices are limited in computing power and memory resources. To overcome these limitations, ongoing efforts are focused on minimizing power consumption and optimizing hardware performance for effective machine learning (ML) processing on edge and IoT devices. A new avenue of interest in this context is Hyper-Dimensional Computing (HDC) for classification tasks [1], which promises hardware-friendly, parallelizable computations [2], noise, and runtime error resistance ML [3].

The calculations for inference in HDC can be divided into two separate parts: the encoding part, which transforms the input vector into a hyperdimensional space of the order of a few thousand, called Query HyperVector (QHV), and the classification part, which finds the most similar Class HyperVectors (CHV) with the generated QHV. The CHVs are generated during the training of the HDC model by bundling the HVs of each class independently over all inputs of the training dataset. The calculations in HDC are mostly bitwise XOR operations instead of the multipliers used in most neural networks, which significantly reduces hardware complexity and improves energy efficiency. However, a major obstacle in most advanced hardware implementations of HDC [4]–[6] is the significant memory requirements for accessing the fixed HyperVector (HV) parameters that are used during encoding. Storing large constant HVs is not only unsuitable for resource-constrained edge devices with limited storage but also imposes a noticeable energy cost for devices with limited batteries.

This paper proposes a novel, hardware-efficient encoding technique that significantly reduces the required memory and energy footprint of the encoder by enabling the generation of encoding parameters on-the-fly at runtime and replacing the large memory blocks with hardware-friendly logic to generate the parameters with negligible impact on accuracy.

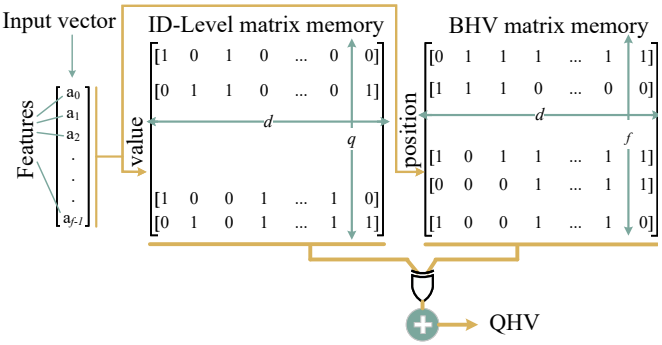

Fig. 1. The base-level encoding approach.

II. Proposed Encoding Technique: OTFGEncoder

Our lightweight HDC encoding technique, On-the-Fly Generation Encoding (OTFGEncoder), generates encoding parameters on-the-fly at runtime. This method is based on the well-known base-level encoding approach (illustrated in Fig. 1), which requires two large memories for storing the Base HyperVector (BHV) matrix and the ID-level matrix. For each input feature, the value is used as the index for the ID-level matrix, and the position of the feature is used as the index for the BHV matrix. The output of these two memories is then XORed. All input features must go through this step and be added together to create the QHV. In this work, the memory units are replaced by generator circuits that can generate the values at runtime. In this section, the OTFGEncoder is explained in two separate subsections.

A. ID-level encoding

In the base-level encoding method, every possible value of the input features must be transferred to the hypervector space before further computations can be performed. We introduce an adaptive system called *ID-Gen* that generates the level hypervectors on-the-fly instead of retrieving them from memory. To form ID-level hypervectors and maintain high classification accuracy, values close to each other are translated to HVs with a higher similarity and vice versa.

Our proposed ID-Gen module, shown in Fig. 2-a, is parameterized based on the dimension size d of the hypervectors and the range n of the quantized features, where features x

979-8-3503-4859-0/DATE24/© 2024 EDAA

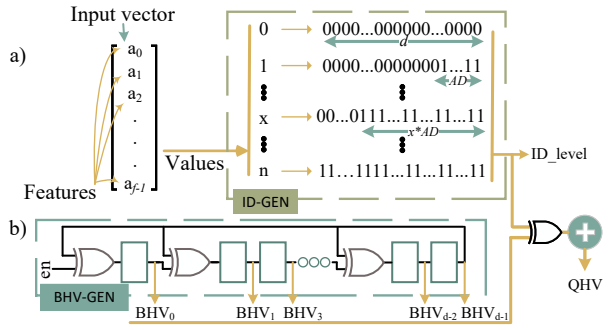

Fig. 2. Overview of OTFGEncode technique, where ID-level and BHV matrices are replaced by low-overhead hardware-friendly generators

can take values $x \in [0, n)$: We first choose a step size AD such that $AD = \frac{d}{n}$. Then a hypervector is generated for a given value x as a sequence of $d - (x \times AD)$ zeros followed by a sequence of $x \times AD$ ones. In this scenario, a succession of zeros is considered for the lowest potential value of the input features, and the number of ones in the HV increases for higher feature values. In this approach, the dissimilarity of the HVs correlates directly with the difference between the values of the input vector without the need for memory.

B. Generating BHVs

To generate a suitable BHV matrix for HDC, the orthogonality of the rows in the matrix and the repeatability of the entire matrix for each input must be taken into account. The proposed alternative hardware to replace the BHV matrix memory takes advantage of cyclic architecture circuits that can generate repetitive data sequences with low hardware overhead and is called BHV-Gen. BHV-Gen combines two flexible cyclic architectures: Linear Feedback Shift Register (LFSR) and Multiple Input Signature Register (MISR). The LFSR is a chain of registers that are XORed with the output of the last register and can structure any arbitrary polynomial, e.g., $x^d + x^{d-3} + ... + x^0$. The LFSR can generate several sequences with different seed values and different polynomials. However, the LFSR generates a one-bit output in each clock cycle. The MISR concept, on the other hand, supports parallel output vectors depending on a series of parallel inputs. Our proposed cyclic BHV-Gen structure combines these concepts as shown in Fig. 2-b, where *d* number of registers is connected in a chain and the output of all registers constructs a single row of the BHV matrix. The intuition is that the combination of shifting and feedback loops inherently generates pseudorandom vectors that are also orthogonal. We also confirm this intuition empirically by evaluating more than 10 000 different LFSR configurations and seed values with respect to the orthogonality of the generated BHVs. Considering orthogonality as the dot product of bipolar vectors ($0 \rightarrow -1$, $1 \rightarrow 1$), the ideal value would be 0 between all BHVs. In our evaluation, the average orthogonality would be about 0.04 for simple cyclic shift registers as used in [7], but about 0.02 when using our LFSR-like structure.

The BHV-Gen structure generates the BHVs row by row in each clock cycle. We note that this does not limit the throughput, as the whole BHV matrix cannot be loaded from memory within a single clock cycle either. However, multiple rows could be generated simultaneously if multiple LFSRs with the same configuration and different initial seeds are used.

III. Evaluation

To illustrate the impact of the encoding technique on accuracy, we modified an open-source library called TorchHD [8] to include support for OTFGEncoder. We then performed a comparison between this modified version, the base model, and another HDC work that also uses TorchHD as their training library. This comparison was performed for different types of classifications as listed in Table I. The results demonstrate that the memory storage for the encoding parameters can be generated by a hardware-friendly on-the-fly mechanism with negligible impact on the accuracy of the classification compared to the baseline.

IV. Acknowledgement

This work was supported by funding from the pilot program Core Informatics of the Helmholtz Association (HGF).

References

[1] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, 2009.

[2] S. Datta *et al.*, "A programmable hyper-dimensional processor architecture for human-centric iot," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, 2019.

[3] H. Li *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2016, pp. 16–1.

[4] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE international conference on rebooting computing (ICRC)*. IEEE, 2017.

[5] ——, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019.

[6] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 international symposium on low power electronics and design*, 2016.

[7] B. Khaleghi *et al.*, "tiny-HD: Ultra-efficient hyperdimensional computing engine for IoT applications," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021.

[8] M. Heddes *et al.*, "Torchhd: An open-source python library to support hyperdimensional computing research," *arXiv preprint arXiv:2205.09208*, 2022.

[9] P. Vergés *et al.*, "HDCC: A hyperdimensional computing compiler for classification on embedded systems and high-performance computing," *arXiv preprint arXiv:2304.12398*, 2023.

TABLE I
IMPACT ON ACCURACY FOR DIFFERENT DATASETS.

Data set	*OTFGEncoder*	*TorchHD [8]*	*HDCC [9]*
Dimension Size	**1000**	1000	10240
MNIST	**82.98%**	82.66%	82.8%
ISOLET	**82.64%**	78.78%	84.8%
EMG	**97,43%**	98,88%	99.3%
GraphHD	**89.47%**	89.47%	-
VoiceHD	**85.18%**	85.05%	-