Hyperdimensional Computing Quantization with Thermometer Codes

Caio Vieira*, Jeronimo Castrillon†‡, Antonio Carlos Schneider Beck*
*Institute of Informatics, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
†Center for Advancing Electronics Dresden, TU Dresden, Dresden, Germany
‡Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), Dresden, Germany
*{cravieira, caco}@inf.ufrgs.br, †‡jeronimo.castrillon@tu-dresden.de

***Abstract*—Hyperdimensional computing (HDC) is an emerging brain-inspired machine learning framework that allows robust and low power computing by exploiting properties of high dimensional vectors. The framework defines a small set of abstract operations, with implementations that vary according to the concrete HDC model used. The Multiply Add Permute (MAP) model, for instance, provides higher accuracy when compared to the Binary Spatter Code (BSC) at the cost of more complex operations. On the other hand, BSC is the preferred model when deploying to embedded devices since it relies on simple bitwise operations. Previous works have proposed quantization methods to convert MAP models into BSC models to provide good performance with minimum accuracy loss. In this work, we introduce TQHD, a novel quantization technique which encodes MAP vectors into thermometer encoded binary vectors. We observe that values in MAP vectors have a regular normal distribution regarding a vector's standard deviation. We split this value range into multiple intervals and map each of them to a thermometer encoded binary word. The quantized vector can be computed using bitwise operations and provide low accuracy loss when compared to the unquantized model.**

***Index Terms*—Brain-inspired computing, hyperdimensional computing (HDC), quantization.**

I. Introduction

Hyperdimensional Computing (HDC) has emerged as a promising solution to allow machine learning in simple hardware due to its cheap computation requirements [1], [2]. Its application as a lightweight classifier has been investigated in several different fields, such as voice [3] and handwritten digit [4] recognition, graphs classification [5], [6], face detection [7], epilepsy detection [8], and robotics [9], [10]. HDC is a neuroinspired computing paradigm that mimics the human brain by computing on hyperdimensional vectors (HV). Each HV represents a piece of information in the application domain. In language recognition, for instance, a HV can represent a single letter, a word, or an entire text. The information contained in HVs is retrieved by similarity measurement. The more similar two vectors are, the higher is the probability they represent the same piece of information. This enables the core idea behind HDC classification, which relies on training HVs to represent a whole class of the modelled problem. For each prediction, the input data is encoded into a HV and used to perform the similarity search on a associative memory (AM). The AM is a special structure in HDC applications that stores all trained class vectors, and similarity search is the act of retrieving the most similar vector for a given input [11].

To compute on HVs, HDC defines a set of operations, whose implementation vary accordingly to the underlying HDC model implemented [12]. The Binary Spatter Code (BSC) [13] implements HDC in binary presenting values in the set $\{0, 1\}$, which occur with equal probability in a HV, and is the preferred model when deploying HDC applications onto resource-constrained hardware. The model compute using cheap bitwise operations and is capable of providing a satisfiable accuracy. Other models, such as Multiply Add Permute (MAP) [14] use integer or floating point vectors initialized with a random and uniform distribution in the set $\{-1, 1\}$. This model is capable of providing better accuracy and requires smaller vectors than BSC. Although the two models can be used for the same application, the simpler operations in BSC have made them attractive for implementations in emerging hardware, e.g., in in-memory accelerators [15], [16].

To bridge the accuracy gap, previous works have investigated model quantization as an alternative to provide lower computation costs with minimal accuracy loss [17]–[19]. The naive *sign quantization* method to turn a MAP model into a BSC one maps positive values to 1 and negative values to 0. However, this approach incurs in high accuracy loss since each dimension in a vector can quickly become distant from its initial $\{-1, 1\}$ values after some operations, i.e., additions and multiplications. Therefore, this approach can erase valuable information from the trained vectors. More elaborate quantization approaches have been proposed that expose different trade-offs. QuantHD [17] and ReHD [19] use sign quantization and cope with the accuracy loss by several rounds of retraining. The main advantage of this type of quantization is that the quantized vector maintain the same number of dimensions as the original model. On the other hand, PQ-HDC [18] eliminates the retraining step by expanding each dimension of a MAP vector into multiple dimensions in its BSC counterpart.

In this work, we introduce a novel quantization technique named TQHD. We propose to quantize MAP vectors to thermometer encoded binary vectors. The idea is to expand each dimension of the original vector into multiple thermometer encoded binary bits. Thus, the binarized vectors do not belong to the BSC class since they do not feature the same probability of 0s and 1s. This approach can be considered as relaxation to the criteria followed by previous works since they quantize to BSC. Still, the binarized vector can be computed as any

BSC vector when performing similarity search. Our approach is based on the insight that normalized MAP vectors are in a known range $[-1, 1]$ and symmetrically distributed around 0. This observation allows us to divide the interval into segments and map each segment to a thermometer encoded binary word. The transformation can be applied to a trained MAP model and does not require training after quantization. We evaluate our method in several HDC applications.

II. Related Work

Quantizing machine learning models has became an important alternative to lower the high computational costs of complex neural networks [20]. The methods used in quantization can be split into two main categories: Post-Training Quantization (PTQ) and Quantization-Aware-Training (QAT). In PTQ, the model is quantized after training without further retraining steps, which usually reduces the accuracy. QAT achieves higher accuracy than PQT but requires much more computational effort in the training.

Quantization in HDC has a broader perspective since it includes techniques that maintain the same model class, only simplifying the data types used in computation and model-to-model transformations. In the latter, the technique quantizes the original MAP values to bits, allowing the designer to use BSC operations instead of their original MAP counterpart.

QAT: QuantHD [17] claims to be the first to investigate quantization in HDC. The goal is to transform a MAP AM into a BSC AM. The authors' approach can be divided in three steps. 1) The model is trained as any other MAP model. 2) The quantization occurs after training, and two different quantization targets are evaluated: Binary and Ternary. Binary uses the sign function to map originally positive values to 1 and negative values to 0, whereas ternary quantization maps the values to three possibilities $\{-1, 0, 1\}$. 3) To ameliorate the accuracy penalty, the authors do a training after quantization step that can take several rounds. The idea is to run predictions using the quantized AM and updated the original MAP AM when a misprediction occurs. In the next round, the updated MAP AM is quantized again and the procedure repeats. In ReHD [19], the authors use BRIC [21] to optimize the encoding of multiple HDC applications and show that AM search becomes the most resource consuming step in a HDC application. The authors tackle this problem by using a similar retraining technique as proposed in QuantHD, but now exploring a n-bit quantization instead. Moreover, the authors also show how it is possible to prune insignificant dimensions at execution time based on the variance of the values in a vectors. The main advantage of both works is not increasing the dimension of the quantized models.

PTQ: PQ-HDC [18] introduces a novel quantization method that eliminates retraining by quantizing each vector in the MAP AM to multiple BSC vectors. The idea is to shrink the quantization loss progressively by approximating the original AM vector with multiple BSC vectors created by sign quantization. The similarity is measured using the pseudo-hamming distance, which still uses the regular hamming distance, but now the distance to each BSC vector is multiplied by the weight of that vector regarding how it approximates the original MAP vector.

Our work: We introduce TQHD, a novel way to quantize MAP models. Our work can be classified as PTQ, since it eliminates retraining at the cost of increasing the number of dimensions of the quantized model. The final product of our quantization is a binary vector encoded with thermometer words. We stress that this vector is not strictly BSC since it does not have an equal amount of 0s and 1s. Still, hamming distance can be used to perform similarity search.

III. Background

Typical HDC applications work in a pipeline fashion and can be divided in three steps: Map, encode, and search. The first step is to map the input features from the dataset to HVs. These vector are known as *seed* vectors since they are the first vectors used in the model. There are multiple embedding techniques to perform the data mapping to seed vectors and the model implementer must choose the best suitable to the data features depending on the problem being modelled [1]. After mapping the input features and retrieving the appropriate seed vectors, these are used as input to the encoding stage. This stage is responsible for the symbolic reasoning of the model and employs three different operations: Bundle, Binding, and Permute. Each operation has its own semantic, which does not vary depending on the underlying HDC model being used. Bundle receives multiple vectors as input and compute a new vector similar to its operands. On the other hand, binding also takes multiple operands, but computes a new vector dissimilar to them. Permute receives just one operand and computes a new vector dissimilar to the original. The operations can be implemented differently depending on the HDC model. Bundle, Binding, and Permute are implemented using bitwise majority, exclusive or, and bitshift in BSC, but use element wise addition, element wise multiplication and rotation in MAP.

The result of the encoding stage is a vector also known as **query vector**. In classification, the final step is to use the query as search key in the AM. The AM is a structure that holds all trained vectors for the application. The final classification of the input data sample corresponds to the most similar class found in the AM. As occurs with the underlying implementation of operations, the implementation of similarity measurement also varies with the HDC model used. MAP uses the cosine as similarity function. Given two vectors A and B, the result of $cos(A, B)$ tends to 1 if the vectors are similar, 0 if they are orthogonal, and -1 for opposite vectors, or strongly dissimilar. On the other hand, BSC vectors use the hamming distance to compute the similarity between vectors. In this case, the hamming distance of two vectors $H(A, B)$ tends to 0 if the vectors are similar, $D/2$ for orthogonality, and D for strongly dissimilarity.

Training in HDC occurs similarly to prediction and reuses the first two stages, mapping and encode. The objective of training is to produce the class vectors used in prediction. To accomplish that, all query vectors belonging to the same class are accumulated using the bundle operation since it produces

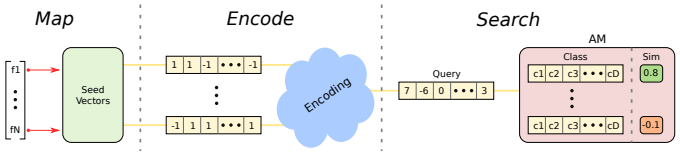

Fig. 1. Typical workflow in an HDC application.

a new vector that is similar to its operands. This is the most common learning technique used in HDC and is known as *Centroid*. Other learning techniques can include solutions to deal with problems such as imbalanced datasets [22] and unsupervised learning [23].

IV. Transformation

A. Basics

Since we propose to perform the vector search in binary, our main challenge is to maintain the similarities between transformed MAP vectors in the binary space. To accomplish this, it is necessary to translate the cosine of vectors to the Hamming distance. Since the two functions are different, a first step is to establish a common ground to compare both metrics. We introduce the $sim(A, B)$ function as a general similarity measure of two random vectors. The function tends to 1 if both vectors are similar and to 0 if not. To write the MAP similarity in terms of the new proposed function, the output range of cosine has to be mapped from $[-1, 1]$ to $[0, 1]$:

$$sim(A, B) = \frac{cos(A, B) + 1}{2}$$

If vectors A and B are normalized, then it is possible to measure the similarity by only computing the dot product of vectors. In this case, $sim(A, B)$ can be written as:

$$sim(A, B) = \frac{dot(A, B) + 1}{2} \tag{1}$$

On the other hand, binary similarity can be written as:

$$sim(A, B) = \frac{D - H(A, B)}{D} \tag{2}$$

We seek a transformation that takes two MAP vectors A and C and transform them to binary vectors A_B and C_B such that $sim(A, C) \sim sim(A_B, C_B)$. A simple approach is to take advantage of the normalized vectors' values being in the interval $[-1, 1]$ and use a sign function to map all negative values to 0 and positive values to 1. This approach is capable of maintaining the sign similarity contribution between MAP values also in its binarized form. For instance, consider the i^{th} element in A and C. If A_i and C_i have equal sign, then their contribution in dot product is positive, increasing the final similarity. On the other hand, if they present different signs, then the contribution in the dot product result is negative, decreasing the final similarity. However, this approach ignores completely the magnitude of each element, which plays an import role in the final dot product value, and thus, similarity computation. This leads to high accuracy losses after model conversion.

To solve this problem, it is necessary to keep the magnitude information after model conversion. However, binary models are only capable of storing one piece of information per dimension. We propose to increase the vector dimension D of the transformed vector, so that extra dimensions can encode the magnitude information. Despite the increased number of dimensions, binary vectors still offer cheaper computation compared to the MAP model. This approach introduces the three main problems: 1) The number of bits B that each dimension of the MAP vector expands to, 2) the number of intervals I that the range of values of normalized vectors are quantized to, and 3) how to properly encode sign and magnitude information in a binary vector.

The main advantage of normalized vectors is that their values are in a known range of $[-1, 1]$. Thus, it is possible to split this interval in I equally distanced points we name as *quantization poles*. We can define the poles as a sorted array $P = [P_1, P_2, \ldots, P_I]$, where I is the number of intervals, P_1 is the first pole and P_I is the last such that P_1 and P_I are the smallest and the biggest values, respectively. Each value in the normalized input vector is assigned to the closest quantization pole. One important requirement is to choose the quantization poles' values in a way that do not allow a value in the original MAP vector to change its sign. To illustrate this problem, consider the value -0.02 and two poles $P_i = -0.1$ and $P_{i+1} = 0.04$. In this case, the value is quantized to P_{i+1} instead of P_i, changing its sign. This problem would be solved if $P_{i+1} = -P_i$. In that case, -0.02 would be correctly mapped to P_i, i.e., -0.1. Thus, we find that the values in P must be symmetric around 0 to avoid *sign flipping* after quantization. As a direct consequence of this decision, $P_1 = -P_I$. Another important observation is that if I is an odd value, then the median of P is at 0.

Next, the values in P need to be encoded in a meaningful binary word for Hamming distance. Since the array P is sorted, the binary encoding of value P_i must be similar to value P_{i+1}. Another consideration is that values P_i and P_I must be as dissimilar as possible. Both requirements are satisfied by the thermometer encoding [24]. Starting from an initial binary vector with B bits named as T_1, the encoding of the following values occurs by flipping one bit at each iteration. Therefore, the number of possible encodings is $B + 1$. For instance, if we choose $B = 4$ and start with a binary number of 4 0s, then it is possible to build 5 different combinations: $T_1 = 0000$, $T_2 = 1000$, ..., $T_5 = 1111$.

It is necessary to map the quantization poles, array P, to binary encodings. Each pole must be mapped to a single thermometer word, thus, $I \leq B + 1$. If $I = B + 1$, the mapping is straightforward, since it is possible to create a map $M = \{(P_1, T_1), (P_2, T_2), ..., (P_I, T_{B+1})\}$. If $I < B + 1$, then there are more binary words available than quantization poles. In this case, we choose to amplify the distance for values with different signs, as is the case in dot product. To achieve this, we map the first half of values in P to the first encodings T available and the second half of P to the last words in T. This mapping maximizes the distance between values with different signs while still considering the magnitude difference between values with the same signs.

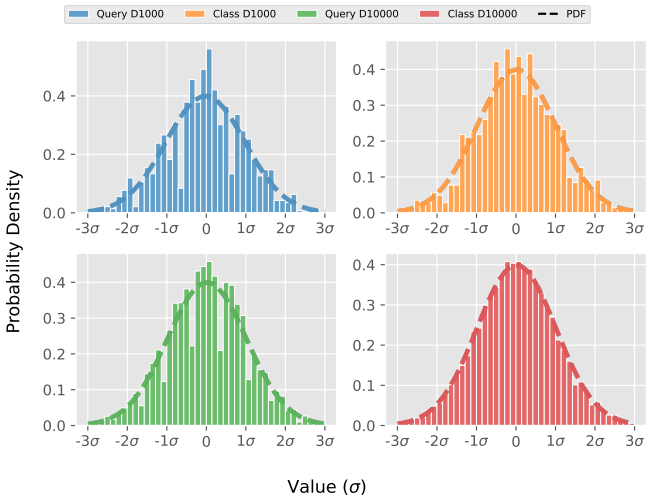

Fig. 2. Probability density of query and trained class vectors of an HDC model scaled to vectors standard deviation (σ) for different values of dimensions (D).

B. Choosing Quantization Values

So far, we have only discussed what are quantization poles and how they should behave. However, we did not discuss how to choose the values of quantization poles P. Since the values must be equally spaced between P_1 and P_I and $P_1 = -P_I$, the main challenge is to choose the initial value P_1. Even though the input vector is normalized, its values can be anywhere in the range $[-1, 1]$. Thus, it is necessary to understand how the values present in normalized vectors are distributed in HDC models.

Figure 2 shows histograms of the values in normalized query and class vectors scaled to their standard deviation (σ) for vectors of different dimensions D. All histograms can be approximated by a Gaussian Probability Density Function (PDF). This distribution is insightful and allows some observations. 1) Although a normalized vector can present values in $[-1, 1]$, usually the values are in a much narrower range that can be related to its standard deviation. In the case depicted, all values fit in the range $[-3\sigma, 3\sigma]$. Thus, the quantization poles can be chosen according to the standard deviation. 2) Class vectors approximate more to the normal distribution than query vectors. The reason behind this is that class vectors are formed by the bundle of multiple query vectors, which in turn are created by operations upon seed vectors. Seed vectors are created with 50% probability of -1 and 1, i.e., its mean is 0 and its values are $\{-\sigma, \sigma\}$. Thus, seed vectors value distribution does not resemble a normal function. However, we observe that as more hypervectors are computed, the more the normalized result values resemble a normal distribution. 3) Even though all operations available in the MAP model (multiplication, addition, and permutation) preserve the mean but change the standard deviation, after normalization, all hypervectors tend to present the same σ. Thus, normalized query and class vectors of a model tend to present values in the same range related to σ, and σ only changes when varying the number of dimensions D in a model. Since D is fixed in a HDC model and the values of normalized query and class vectors are in a similar value range, we can safely quantize both vectors using the same quantization poles P and then measure their similarity in binary.

Finally, we choose the initial quantization pole P_1 based on a value of σ. As σ only varies with D, and this is fixed in a model, the quantization poles can be created at startup and remain the same throughout execution. In Section V-B, we perform experiments to show how the accuracy loss inflicted by the technique vary compared to the unquantized model.

C. Quantization Pipeline

TQHD can be applied to the backend of any MAP model by swapping the similarity search stage to TQHD similarity search. Figure 3 depicts the integration of TQHD pipeline to the frontend of a MAP model in a inference scenario. First, TQHD normalizes the query vector. Next, it searches the closest quantization pole for each dimension in the normalized vector. Then, each dimension is expanded to a binary word. Finally, the binarized vector is used as query vector to the Thermometer Quantized AM (TQAM) created in the training phase. Our technique does not change the training phase, hence the model can be trained as any other MAP model. After the training is complete, each vector in the AM is quantized with the same method used to quantize query vectors in inference to create the TQAM.

V. Results

A. Experimental Setup

Our custom HDC model is implemented in Python using TorchHD [25], an HDC library built upon Pytorch [26]. We evaluate our proposoal on four HDC applications: Voice recognition using the ISOLET dataset (VoiceHD) [3], digit recognition with MNIST [4], language recognition based on [27], and a hand gesture recognition based on electromyography (EMG) signals [28]. Table I summarizes the datasets used for each model. The dataset used to evaluate the language model consists of text samples of 21 european languages. Since each sample can have a different size, we ensure that all strings have 128 characters by trimming longer strings and repeating smaller strings. The dataset used in EMG is based on data collected from 5 different subjects. In our EMG evaluations, we always consider the entire dataset by averaging the accuracy obtained in all 5 subjects. Since our main goal is to quantize a model with the minimum accuracy loss, our results mainly show how good the quantization can be. Thus, we choose the accuracy loss in percentage points as our main comparison metric. We execute all experiments using 20 different seeds to evaluate how randomness affects the models. We consider each trained model with a seed s as the baseline of the same quantized model when measuring the accuracy loss of quantization.

B. Quantization Parameters

This section presents experiments to allow for a better understanding on how the quantization parameters (intervals I, bits B, and choice of quantization poles P) affect the accuracy of quantized models.

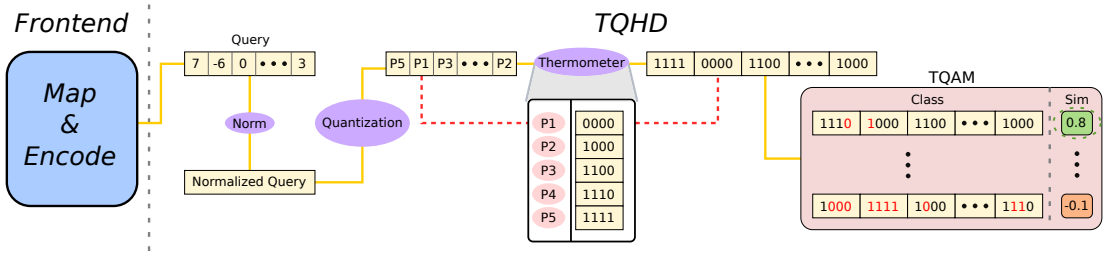

Fig. 3. TQHD pipeline attached to the frontend of a MAP pipeline considering $I = 5$ and $B = 4$.

TABLE I
SUMMARY OF DATASETS FOR EACH MODEL (n: FEATURE SIZE, K: NUMBER OF CLASSES).

Model	n	K	Data Size	Train Size	Test Size
voicehd [3]	617	26	39MB	6,238	1,559
mnist [4]	784	10	53MB	60K	10K
language [27]	128	21	26MB	210K	21K
emg [28]	4	5	48MB	1,619	695

1) Intervals and Bits: In the first experiment, we fix the value of the first quantization pole P_1 to σ[1] and variate the parameters I and B. Our goal is to assess how many bits and intervals are necessary to approximate the original accuracy of a MAP model trained with floating point and $D = 1000$. Figure 4 shows the accuracy loss for each evaluated application for different values of I and B. The models evaluated are arranged in ascending order of B and I, respectively. We evaluate all cases where $I = B+1$. However, due to the symmetry requirements of the quantization poles and thermometer words discussed in Section IV-A, it is not possible to evaluate all possible values of I for a given B. For instance, it is not possible to evaluate the system $I3B3$ because the number of entries in the thermometer table is 4, which must be mapped to 3 intervals, resulting in a asymmetric mapping. As expected, increasing the number of bits tends to improve the accuracy of all models since it allows for better quantization resolution. However, values of $I < B+1$ always lead to worse results. It is important to notice that changing I while fixing B does not affect the quantization cost since each MAP dimension is going to still be expanded to the same amount of bits. Thus, we conclude that it is better to choose the maximum value of I to a given value of B for minor accuracy loss.

Some of the models evaluated present negative accuracy loss, as is noticeable for EMG application. This means that for a given model trained with one seed s, its quantized version was capable of providing a better classification accuracy. This happens because our technique tries to mimic the dot product behavior using Hamming distance. Thus, this can lead to errors that increase the accuracy of the quantized models. Our approach is substantially different, for instance, from QAT techniques that first quantize the AM and then try to recover

[1]In Section IV we defined P as a sorted and symmetric array. Thus P_1 must be a negative value and could never be σ. We relax the definition when discussing results for the sake of simplicity.

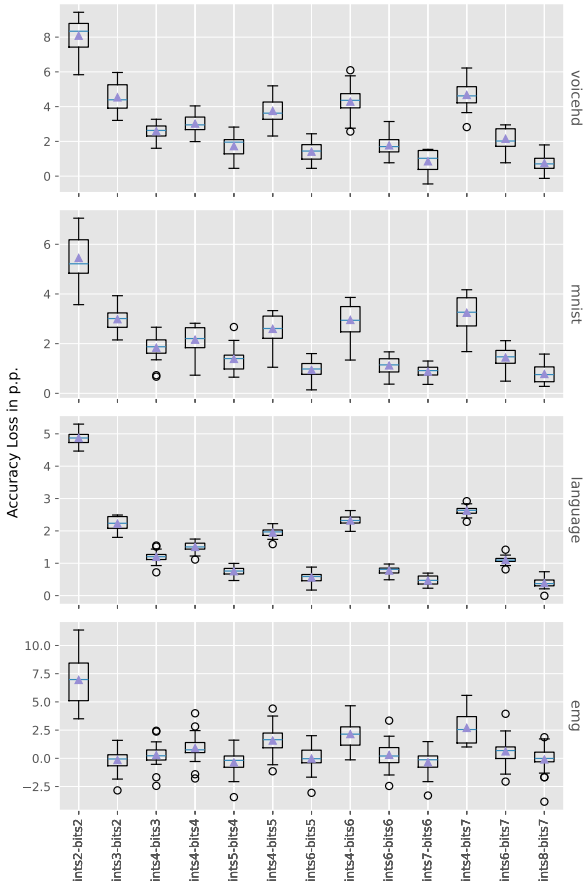

Fig. 4. Accuracy difference to the unquantized model in percentage points for each evaluated application using different intervals (I) and bits (B) quantization parameters.

the accuracy loss by successive rounds of retraining, and also from PQ-HDC since it attempts to approximate the original MAP vector by using multiple weighted quantized vectors. In the case of EMG, configurations that have more resources, i.e., higher B and I, tend to be more stable towards 0 accuracy loss.

2) Quantization Poles: In Section IV-B, we discussed how the quantization poles must be related to σ. The following experiment sheds light into how varying the hyperparameter P_1 affects the accuracy loss. To realize this, we train the applications using $D = 1000$ and compare the accuracy loss when quantizing using different values of B from 2 to 8, and $I = B+1$ since it provides the best accuracy. Figure 5 shows

the mean accuracy loss when varying P_1 in the range $[0.1, 2]$ in steps of 0.1. We also include the mean accuracy loss of sign quantization (SQ) in the plots to allow the comparison with TQHD. In general, all TQHD plots present a hammock curve since there are high accuracy losses towards the end of the intervals evaluated and lower accuracy losses towards the middle. As the choice of P_1 gets closer to 0, the accuracy loss of TQHD tends to be the same as the sign quantization. This is expected because the more the quantization poles P_1 and P_I are close to 0, the more values in the quantized vectors are mapped to the extremes of the quantization table, i.e., positive values tend to be mapped to only 1s while negative values are mapped to all 0s. As the value of P_1 tends to larger multiples of σ, the accuracy loss tends to increase. This is because TQHD loses its capacity to better distinguish the majority of the samples, since they occur at lower values of σ. Even though the choice of the optimal hyperparameter P_1 may require to sweep several values of σ, we observe that the best accuracy loss lies generally around $1 \cdot \sigma$ for all applications evaluated, with minimal accuracy loss variation between the best accuracy loss achievable and $P_1 = \sigma$.

Another important conclusion we draw from this experiment is that higher values of B not only provide better accuracy results, but also diminishes the accuracy loss variation of caused by P_1.

C. Scalability

Finally, we evaluate TQHD regarding scalability with different values of D to assess how the quantization affects larger models. Figure 6 shows the accuracy loss for TQHD and SQ when varying D in the range $[1K, 10K]$ in steps of $1K$. Based on the results discussed so far, we choose the max value of I possible and $P_1 = \sigma$. For all applications and values of D, TQHD features a low accuracy loss. SQ display low accuracy loss for voicehd, mnist and language, but only for high values of D. SQ does not perform well in emg. Based on this results, TQHD seems to cover more applications than the sign quantization.

Another advantage of TQHD is its capacity to provide good accuracy also for low values of D as occurs in voicehd, mnist, and language. One can progressively improve the accuracy by increasing the quantization resolution, i.e., the number of bits B. For all scenarios evaluated, TQHD can almost flawlessly mimic the accuracy of a MAP model if provided with enough resources.

VI. Conclusion

In this paper, we proposed TQHD, a novel method to quantize MAP models to binary that can provide negligible accuracy loss. Our technique attempts to mimic the dot product in binary by using thermometer encoding. Thus, we relax the criteria of previous works since we do not quantize to BSC. TQHD is only possible due to distribution of values in normalized HVs, which we showed empirically to follow a zero-mean normal distribution. Based on this, we divide the range of possible values in segments related to the standard deviation, and map the values close to each segment to a

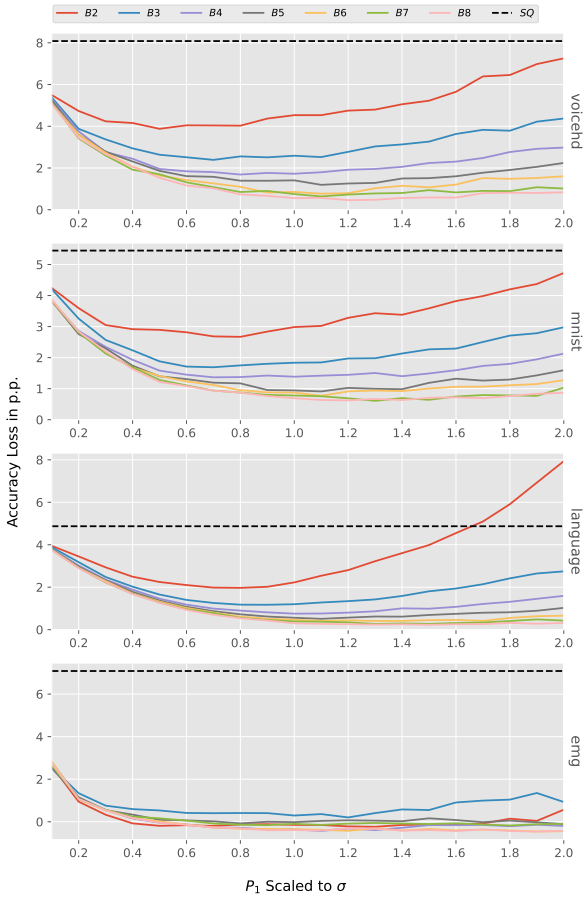

Fig. 5. Accuracy loss in percentage points when varying the choice of P_1 for different number of bits B. SQ depicts the mean accuracy loss of sign quantization.

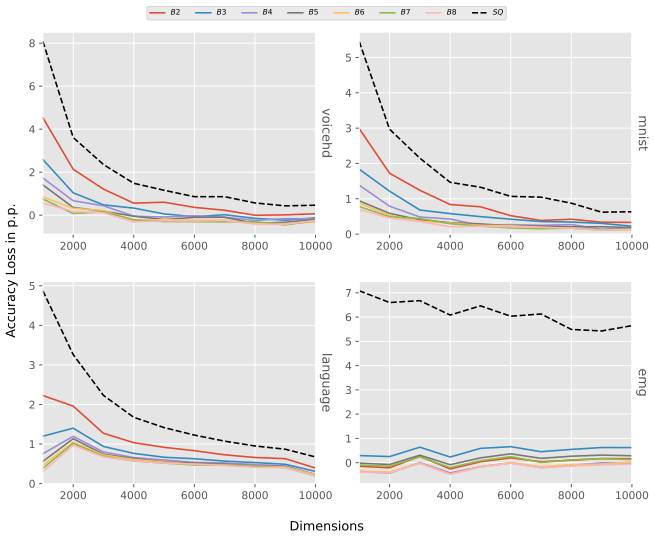

Fig. 6. Accuracy loss in percentage points when varying D in the original MAP model.

thermometer binary word. In our evaluation, we show how the quantization parameters I, B, and P affect the quantization accuracy and provide guidance to choose these parameters. Finally, we show how our technique performs for HDC in general by quantizing MAP models post-training with different sizes of D. We show that our technique can perform better than the naive sign quantization in all cases.

ACKNOWLEDGMENT

This study was financed in part by the CAPES - Finance Code 001, FAPERGS and CNPq.

REFERENCES

[1] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, and F. T. Sommer, "Vector Symbolic Architectures as a Computing Framework for Emerging Hardware," *Proceedings of the IEEE*, vol. 110, pp. 1538–1571, Oct. 2022.

[2] C.-Y. Chang, Y.-C. Chuang, C.-T. Huang, and A.-Y. Wu, "Recent Progress and Development of Hyperdimensional Computing (HDC) for Edge Intelligence," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, pp. 119–136, Mar. 2023.

[3] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, Nov. 2017.

[4] E. Hassan, Y. Halawani, B. Mohammad, and H. Saleh, "Hyper-Dimensional Computing Challenges and Opportunities for AI Applications," *IEEE Access*, pp. 1–1, 2021.

[5] I. Nunes, M. Heddes, T. Givargis, A. Nicolau, and A. Veidenbaum, "GraphHD: Efficient graph classification using hyperdimensional computing," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1485–1490, Mar. 2022.

[6] P. Poduval, H. Alimohamadi, A. Zakeri, F. Imani, M. H. Najafi, T. Givargis, and M. Imani, "GrapHD: Graph-Based Hyperdimensional Memorization for Brain-Like Cognitive Learning," *Frontiers in Neuroscience*, vol. 16, 2022.

[7] M. Imani, A. Zakeri, H. Chen, T. Kim, P. Poduval, H. Lee, Y. Kim, E. Sadredini, and F. Imani, "Neural computation for robust and holographic face detection," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, (New York, NY, USA), pp. 31–36, Association for Computing Machinery, Aug. 2022.

[8] U. Pale, T. Teijeiro, and D. Atienza, "Combining General and Personalized Models for Epilepsy Detection with Hyperdimensional Computing," Mar. 2023.

[9] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, p. eaaw6736, May 2019.

[10] P. Neubert, S. Schubert, and P. Protzel, "An Introduction to Hyperdimensional Computing for Robotics," *KI - Künstliche Intelligenz*, vol. 33, pp. 319–330, Dec. 2019.

[11] P. Kanerva, "Computing with High-Dimensional Vectors," *IEEE Design & Test*, vol. 36, pp. 7–14, June 2019.

[12] K. Schlegel, P. Neubert, and P. Protzel, "A comparison of vector symbolic architectures," *Artificial Intelligence Review*, vol. 55, pp. 4523–4555, Aug. 2022.

[13] P. Kanerva, "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors," *Cognitive Computation*, vol. 1, pp. 139–159, June 2009.

[14] R. Gayler, "Multiplicative Binding, Representation Operators & Analogy," Jan. 1998.

[15] A. Kazemi, M. M. Sharifi, Z. Zou, M. Niemier, X. S. Hu, and M. Imani, "MIMHD: Accurate and Efficient Hyperdimensional Inference Using Multi-Bit In-Memory Computing," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, July 2021.

[16] A. A. Khan, S. Ollivier, S. Longofono, G. Hempel, J. Castrillon, and A. K. Jones, "Brain-inspired Cognition in Next-generation Racetrack Memories," *ACM Transactions on Embedded Computing Systems*, vol. 21, pp. 79:1–79:28, Dec. 2022.

[17] M. Imani, S. Bosch, S. Datta, S. Ramakrishna, S. Salamat, J. M. Rabaey, and T. Rosing, "QuantHD: A Quantization Framework for Hyperdimensional Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 2268–2278, Oct. 2020.

[18] C.-T. Huang, C.-Y. Chang, Y.-C. Chuang, and A.-Y. A. Wu, "PQ-HDC: Projection-Based Quantization Scheme for Flexible and Efficient Hyperdimensional Computing," in *Artificial Intelligence Applications and Innovations*, IFIP Advances in Information and Communication Technology, (Cham), pp. 425–435, Springer International Publishing, 2021.

[19] J. Morris, R. Fernando, Y. Hao, M. Imani, B. Aksanli, and T. Rosing, "Locality-Based Encoder and Model Quantization for Efficient Hyper-Dimensional Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, pp. 897–907, Apr. 2022.

[20] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization," June 2021.

[21] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "BRIC: Locality-based Encoding for Energy-Efficient Brain-Inspired Hyperdimensional Computing," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2019.

[22] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, "OnlineHD: Robust, Efficient, and Single-Pass Online Learning Using Hyperdimensional System," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 56–61, Feb. 2021.

[23] E. Osipov, S. Kahawala, D. Haputhanthri, T. Kempitiya, D. D. Silva, D. Alahakoon, and D. Kleyko, "Hyperseed: Unsupervised Learning With Vector Symbolic Architectures," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022.

[24] D. A. Rachkovskiy, S. V. Slipchenko, E. M. Kussul, and T. N. Baidyk, "Sparse Binary Distributed Encoding of Scalars," *Journal of Automation and Information Sciences*, vol. 37, no. 6, 2005.

[25] M. Heddes, I. Nunes, P. Vergés, D. Kleyko, D. Abraham, T. Givargis, A. Nicolau, and A. Veidenbaum, "Torchhd: An Open Source Python Library to Support Research on Hyperdimensional Computing and Vector Symbolic Architectures," *Journal of Machine Learning Research*, vol. 24, no. 255, pp. 1–10, 2023.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, no. 721, pp. 8026–8037, Red Hook, NY, USA: Curran Associates Inc., Dec. 2019.

[27] A. Joshi, J. T. Halseth, and P. Kanerva, "Language Geometry Using Random Indexing," in *Quantum Interaction*, Lecture Notes in Computer Science, (Cham), pp. 265–274, Springer International Publishing, 2017.

[28] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, Oct. 2016.