2021 IEEE 39th International Conference on Computer Design (ICCD) | 978-1-6654-3219-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCD53106.2021.00058

Stochastic-HD: Leveraging Stochastic Computing on Hyper-Dimensional Computing

Yilun Hao*, Saransh Gupta*, Justin Morris*†, Behnam Khaleghi*, Baris Aksanli†, and Tajana Rosing*
*University of California San Diego, La Jolla, CA 92093, USA
†San Diego State University, San Diego, CA 92182, USA
{yih301,sgupta,justinmorris,bkhalegh}@ucsd.edu, baksanli@sdsu.edu, tajana@ucsd.edu

***Abstract*—Brain-inspired Hyperdimensional (HD) computing is a novel and efficient computing paradigm which is more hardware-friendly than the traditional machine learning algorithms, however, the latest encoding and similarity checking schemes still require thousands of operations. To further reduce the hardware cost of HD computing, we present Stochastic-HD that combines the simplicity of operations in Stochastic Computing (SC) with the complex task solving capabilities of the latest HD computing algorithms. Stochastic-HD leverages deterministic SC, which uses structured input binary bitstreams instead of the traditional randomly generated bitstreams thus avoids expensive SC components like stochastic number generators. We also propose an in-memory hardware design for Stochastic-HD that exploits its high level of parallelism and robustness to approximation. Our hardware uses in-memory bitwise operations along with associative memory-like operations to enable a fast and energy-efficient implementation. With Stochastic-HD, we were able to reach a comparable accuracy with the Baseline-HD. As compared to the best PIM design for HD [1], Stochastic-HD is also 4.4% more accurate and 43.1× more energy-efficient.**

I. Introduction

Brain-inspired Hyperdimensional (HD) computing is a light-weight computing method to perform cognitive tasks on devices with limited resources [2], [3] such as activity recognition, object recognition, language recognition, and bio-signal classification [4], [5], [6]. HD computing has three main stages, 1) Encoding: mapping data into *hypervectors* (HVs). 2) Training: combining encoded HVs to create a model representing each class. 3) Inference: comparing the incoming sample with the trained model to find the most similar class.

HD has highly parallelizable operations by operating on independent HVs with $10,000$ dimensions. However, during encoding and inference, there are numerous element-wise multiplies and accumulations due to matrix multiplication. When mapped to a processing-in-memory (PIM) architecture, they have to be performed sequentially.

By implementing same inference process using Naive PIM in [7], the XOR operations costs 3.3ns because they are able to work completely in parallel, while the accumulation (which includes the element-wise multiplication) costs 13504ns because they needs to work sequentially as the operations are not simple bitwise operations. These operations cause a clear bottleneck for HD computing in PIM architecture. Although this can be alleviated by significantly increasing the memory size, which costs 97ns, that comes at a $D\times$ increase in area, where $D = 4,000$. Prior work leveraged simple analog PIM memory to implement HD Computing [1] to solve this issue but they lose, on average, 3% of the accuracy of HD Computing because the analog circuits are too aggressive at approximation. To alleviate this issue, we utilize Stochastic Computing (SC) operations, a computing paradigm that uses random bitstreams to convert complex computations to simple bit-wise operations on the streams [8], in their place, which are much more hardware friendly for PIM architectures. Additionally, with SC, we are able to maintain the accuracy of HD better than previous aggressive analog PIM architectures.

In this paper, we leverage SC during the encoding and similarity checking phase of both training and inference stages of HD computing. We propose, Stochastic-HD, which combines HD computing and SC to perform classification tasks in PIM with highly parallel operations. To do this, we utilize deterministic SC [9], which uses a more structured way to represent the bitstreams instead of typical randomly generation thus results in better accuracy. We first turn the encoded class hypervectors into bitstreams. During retraining and inference, we represent each element of the query hypervectors with bitstreams, and then use AND to compare the similarity instead of cosine similarity. With Stochastic-HD, we were able to reach a comparable accuracy with the Baseline-HD. As compared to the best PIM design for HD [1], Stochastic-HD is 4.4% more accurate and 43.1× more energy-efficient.

II. Related Work

A. Hyperdimensional Computing

Prior work applied the idea of HD Computing to different classification problems such as language recognition, speech recognition, face detection, human-computer interaction, and sensor fusion prediction [4], [5], [6]. Prior works also proposed binary encoding to accelerate HD Computing[10], [1], [11].

Prior work has also proposed hardware accelerators for HD Computing such as FPGAs [12], and PIM architectures [13], [1]. Although GPUs and FPGAs provide a suitable degree of parallelism that makes them amenable to machine learning algorithms such as deep neural networks, the complexity of their resources, e.g., floating point units or DSP blocks, is by far beyond HD's requirements, making such devices inefficient for HD. PIM architectures tackle this problem as they are comprised highly parallel arrays with intrinsically non-complex computational capability, which is sufficient for HD operations. Additionally, PIM can eliminate the high cost

978-1-6654-3219-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ICCD53106.2021.00058

data movement in the traditional von Neumann architectures as, in PIM, data resides where computation is performed. Adding a PIM accelerator for HD computing to perform cognitive tasks provides significant speed up over utilizing the on-board CPU and saves energy with less data movement.

B. Stochastic Computing

Prior work has utilized SC [14], [8], [15] as a low-cost computing paradigm which process random bitstreams for implementations of convolutional neural networks [16]. To calculate $p \times q$ for $p, q \in [0, 1]$, SC generates two random independent binary bitstreams, where the probability of a '1' in the first and second bitstream is p and q. For exmaple, the stream 1010011110 represents 0.6 since it consists of six 1's with a total of ten bits. With converting numbers into bitstreams and using a single AND operation to perform multiplication, SC successfully reduced the power and cost of complex but necessary operations such as multiplication in implementation of neural networks. The usage of long bitstreams to represent data also ensures that SC implementations are noise tolerant.

However, when the bitstream length is not long enough, with random arrangements of 1's and 0's, the result of ANDing two bitstreams may not be exact. The random fluctuations resulted from the generation of bitsteams cause the computation of SC to be approximate. The accuracy increases when the bitstream is long enough to better approximate the probability. This significantly reduced the accuracy of SC based implementation.

C. Deterministic Stochastic Computing

To resolve this problem, prior work raised deterministic Stochastic Computing as an algorithm that computes on deterministic bit streams, which could reduce the area, reduce the latency, and produce completely accurate results [9]. By properly structuring input bitstreams, completely accurate results can be produced with no random-fluctuation or correlation errors. For example, by using 100 to represent $\frac{1}{3}$ and 1110 to represent $\frac{3}{4}$, then repeat these two small streams until it pairs every bit from one bitstream with every bit of the other bitstream exactly once(100100100100, 111011101110), to get an accurate result of $\frac{3}{12}$(100000100100) by taking AND.

III. HD with Stochastic Computing

We propose Stochastic-HD, a novel algorithm that takes the advantages of both HD computing and Stochastic computing. Stochastic-HD consists of three main modules: encoding, training, and testing. Our method is designed to reduce cost and power for encoding and similarity checking that happens in both training and testing stages. We propose both HD with deterministic SC, which is mainly used for similarity checking, and sign deterministic SC, which is mainly used for encoding.

A. Encoding

Baseline HD HD computing encoding maps each n dimensional feature vector to a D dimensional binary hypervector. We utilize a random projection encoding presented in [7]. The encoding first generates D dense bipolar vectors with the same dimensionality as original domain, $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_D\}$, where $\mathbf{p}_i \in \{-1, 1\}^n$. Thus, to encode a feature vector into a hypervector, we perform a matrix vector multiplication between the projection matrix and the feature vector using $\mathbf{H} = sign(\mathbf{PF})$, where $sign$ is a sign function which maps the result of the dot product to +1 or -1. However, this encoding process is very computationally expensive as it consists of thousands of arithmetic operations with a high dimension D. As a result, we want to apply SC as a light-weight hardware-friendly encoding algorithm to enable parallelism to solve this.

Stochastic HD Instead of using multiplication, we use logical AND and XOR to accomplish the same calculation. Since the projection matrix $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_D\}$ has $\mathbf{p}_i \in \{-1, 1\}^n$, to apply SC, we use a sign bit of 0 to represent a positive value and 1 to represent a negative value before normal stochastic bitstreams [17]. We do the same for the feature vectors $\mathbf{F} = \{f_1,\ f_2, \ldots,\ f_n\}$.

We generate the stochastic bitstreams first by looking for two fractions to represent the two values we want to multiply with, and then convert them into small bitstreams. As a result, we first cast the input data from float into int. Since in the original process, after matrix multiplication, the value would be cast into +1 or -1, casting the input data from float into int does not cause significant accuracy drop. In addition, since each multiplication of projection matrix and feature vector is composed of multiplication of +1/-1 with f_i, to apply deterministic SC, we could directly use a stream of 1's, of length $max(abs(|f_i|))$ to represent each element in projection matrix, and represent each feature f_i by using $\frac{f_n}{max(abs(f_i))}$ and thus corresponding bitstream. Thus, with all values converted into bitstreams, we were able to take XOR for sign bit and AND for the rest bits to generate accurate results for the original multiplication calculation. After this, we count the number of 1's for both positive sign bits and negative sign bits and do subtraction. If the result is positive, we will cast the result into +1, and otherwise -1. With our implementation, all the multiplications and additions are converted into parallelizable AND or XOR operations, which is more efficient in hardware.

B. Initial Training

During initial training, the model is initialized through element-wise addition of all encoded hypervectors in each existing class. The result of training is k hypervectors each with D dimensions, where k is the number of classes. For example, the i^{th} class hypervector can be computed as: $\mathbf{C_i} = \sum_{\forall j \in class_i} \mathbf{H_j}$. In addition, to limit the data range of the class hypervectors for shorter stochastic bitstream length, we quantized the class hypervector uniformly.

C. Similarity Check

Baseline HD As shown in Figure 1, similarity checking is done by taking cosine similarity of each query and the class hypervector. However, this implementation requires thousands of multiplications and additions, which is significantly computationally expensive. As a result, to enable more hardware-level parallelism, we provide similarity checking with SC.

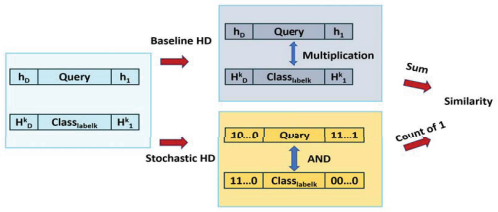


Fig. 1. Comparision of similarity check of Baseline HD and Stochastic HD.

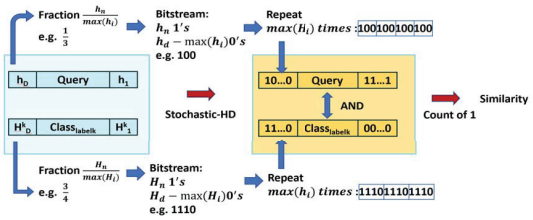


Fig. 2. An overview of similarity checking is done with Stochastic-HD

Stochastic-HD Figure 1 compares the similarity checking of Baseline HD Computing and Stochastic-HD. Instead of cosine similarity, Stochastic-HD uses deterministic SC to generate bitstreams for each data point $\mathbf{h_i}$ in the Query $\mathbf{H_i^k}$ and in the class hypervectors of the encoded model. Then, we put these two bitstreams through an AND gate to produce the result bitstream. To compare the similarity, we simply count the number of $1's$ of each result bitstreams, and the one with most $1's$ represents a larger value, thus closer similarity.

We generate the deterministic stochastic bitstreams first by looking for two fractions that represents the values from query and class hypervector, and then convert them into small bitstreams. Since we are able to get an integer encoding from Stochastic-HD encoding and initial training, we first make all the values in the encoded hypervector to be positive values. Then, the fraction is naturally generated by $\frac{h_n}{max(h_i)}$ and $\frac{H_n}{max(H_i)}$ for the n$^{\text{th}}$ element of query and class hypervector as shown in Figure 2. This means that the bitstream length for query and class hypervector is $max(h_i)$ and $max(H_i)$, and the number of 1's in the bitstream is h_n and H_n. Next, we repeat these small bistreams to form two large two bitstreams until it pairs every bit from one bitstream with every bit of the other bitstream exactly once. This means that we repeat $max(H_i)$ and $max(h_i)$ times for each h_i and H_i respectively.

Now with two bitstreams, we will input them into an AND gate and then count the number of $1's$ to represent the similarity. For example, in Figure 2, to calculate the similarity between Query h and Class hypervector H^k, suppose that $\frac{h_n}{max(h_i)}$ is $\frac{1}{3}$ and $\frac{H_n}{max(H_i)}$ is $\frac{3}{4}$, then the small bitstreams we generated for them is 100 and 1110. Then, repeat them until every bit in the first bitstream matches every bit in the other bitstream exactly once: 100100100100 (repeat $max(H_i)$ = 4 times) and 111011101110 (repeat $max(h_i)$ = 3 times). With all conversion done for all elements in two hypervectors, we will input these two bitstreams into an AND gate array. The number of 1's in the output represent the similarity. And the class with highest similarity would be our prediction.

D. Inference

After initial training, the HD model can now be used for inference. As discussed in similarity checking, Stochastic-HD converts the value to bitstreams and computes the similarity between the *query* hypervector and each class hypervector. It then uses this stochastic similarity to find a class hypervector with the most similarity with the query hypervector.

IV. Stochastic-HD Hardware Design

In this section, we details our implementation of stochastic VMM in Stochastic-HD. We first propose an in-memory search-based binary vector matrix multiplication and then extend it to support input bitstreams in Stochastic-HD. We show how the same memory block design can be used for both Stochastic-HD encoding and inference in memory.

A. Search-based binary VMM:

AnalogPIM is based on the *magnitude* of accumulated current, while in-memory search is based on discharging speed of wordlines. We propose a hybrid approach that provides us with higher precision but without using DACs/ADCs. In associative search, when the data stored in a memory row is the same as the input query, wordline discharges fastest [18]. Moreover, a match between '1' and '1' takes the same time to discharge the memory as a match between '0' and '0'. Also, the discharging is slower for a mismatch as compared to a match [18]. However, in our case, we need to implement an AND operation and differentiate the match operation between (1-1) from the rest of the three combinations (0-0, 0-1, 1-0).

In our design, we only apply '1's at the input query, while leaving the lines corresponding to input '0' floating. This avoids the occurrence of 0-0. Floating the lines results in the current flowing through highly resistive path, which is similar to 1-0 and 0-1 mismatch cases. Although our floating input has a higher resistive path as compared to the mismatch current paths, both are the similar in magnitude as compared to the low resistive path of 1-1 match.

The discharging characteristic of a whole memory row represents the accumulative effect of all the AND operations between the input and stored hypervectors. Hence, the search output of each row represents a dot-product between the input query and the memory row. More the occurrence of '1-1' in the hypervectors, the faster the discharge, the higher the dot product value. This happens for all rows in parallel. The output of the entire memory block is a vector-matrix multiplication. We can increase the precision of the output dot product by sampling the output. Unlike AnalogPIM designs, we sample the output in time domain. For example, to generate a dot product with 4 bits of precision, we use the same search circuits but latch the output at four different time instants.

B. Encoding with Stochastic-HD binary VMM:

We perform search-based binary VMM n times for n-bit input bitstreams. In addition to the AND operation between stochastic bitstreams, we perform XOR operations between the input sign-bits and the stored projection matrix. To perform XOR, we activate the bitlines corresponding to both '0' and '1' for the sign-bit, unlike AND operation where we just activate '1's. We use the accumulators at the memory periphery to add the dot product outputs of the n iterations. Since the final encoded hypervector consists of {-1, 1}, we take the sign bit of the accumulated output. Since each bit of stochastic bitstream has the same significance, we don't require shift-and-accumulate circuits used by traditional analogPIM design and can utilize simple accumulators with low bit-precision.

C. Similarity Check with Stochastic-HD binary VMM:

During the similarity check, we need to take a dot product between an encoded hypervector with input elements {-1, 1} and a class hypervector with multi-bit elements. To implement this in Stochastic-HD PIM, we distribute a class hypervector over multiple memory rows. Each memory row stores 1 bit of the class bitstream for all dimensions. The dot product outputs of all rows corresponding to a class are added together to get the final dot product of the input with that class. The class with the highest dot product is selected as the output class.

V. Experimental Results

A. Experiment Setup

We designed a cycle accurate simulator which emulates the Stochastic-HD functionality. Our simulator pre-stores the randomly generated projection matrix and encodes in-memory using the controller signals. We extract the circuit level characteristics, of basic operations from circuit level simulations for a 45nm CMOS process technology using Cadence Virtuoso and give them as an input to simulator. We use VTEAM memristor model [19] for our memory design simulation with R_{ON} and R_{OFF} of $10k\Omega$ and $10M\Omega$ respectively.

We tested Stochastic-HD encoding and inference performance on CPU using python. For comparison, we utilized a quantized version of HD computing as the baseline HD, which is also implemented in Python [11]. We use $D = 2048$ as the default dimensionality across all tests except when varying the dimensionality. We use an Intel i7 7600 CPU with 16GB memory for our baseline CPU. We use performance counters to measure CPU power and execution time.

We tested our proposed approach on six applications: Speech Recognition (ISOLET) [20], Face Detection (FACE) [21], Activity Recognition (UCIHAR) [22] (PAMAP2) [23], Gesture recognition(EMG) [24], Cardiotocography (CARDIO) [25]

B. Stochastic-HD Accuracy Comparison

Figure 3 compares the classification accuracy of the baseline HD Computing, which uses exact operations with Stochastic-HD, which uses SC operations that are approximate. As the figure shows, Stochastic-HD is comparable in accuracy

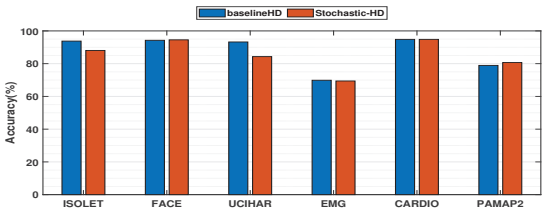
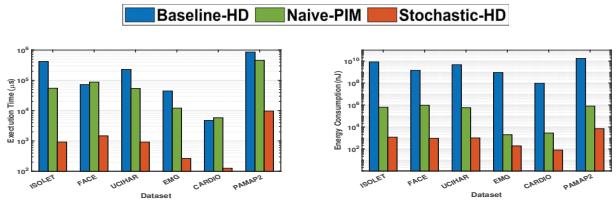


Fig. 3. Accuracy comparison of Stochastic-HD with CPU implementation.

TABLE I
Impact of Dimensionality on Accuracy of Stochastic-HD and Baseline-HD of ISOLET dataset.

Dimension	1,000	3000	5000	7000	9000
Baseline-HD	89.8%	93.8%	94.0%	94.4%	94.2%
Stochasitc-HD	77.9%	88.1%	89.4%	88.7%	91.2%



(a) Execution Time (b) Energy

Fig. 4. Comparison of the inference execution time and energy consumption of Stochastic-HD with CPU and Naive-PIM [7], [1].

to baseline HD. For the datasets ISOLET and UCIHAR, Stochastic-HD is slightly less accurate, losing 5% and 7% in accuracy respectively. These two datasets have more classes that have less separation than the other four datasets. This leads them to be more sensitive to approximate methods like SC, causing Stochastic-HD to be less accurate on them. However, for the other four datasets, Stochastic-HD is able to achieve the same or higher accuracy. For instance, on the PAMAP2 dataset, Stochastic-HD is 3% more accurate. Overall, compared to a baseline with exact computations, Stochastic-HD loses just 1.3% in accuracy on average.

Table I demonstrates the impact on the dimensionality of the HD model on accuracy for both the Baseline-HD implementation and Stochastic-HD on ISOLET dataset. As the table shows, the accuracy very slightly increases as the dimensionality is also increased for both the Baseline-HD and Stochastic-HD. However, the more interesting comparison is with the Baseline-HD with a dimensionality of 1000 vs Stochastic-HD at 5000 dimensions. The Baseline-HD model is able to achieve the same accuracy as Stochastic-HD at this dimensionality and performance scales with dimensionality. However, this would only lead to a $15.9\times$ speedup and $441,889\times$ energy efficiency gain with a naive implementation of the Baseline-HD in PIM. However, with Stochastic-HD we are able to achieve $174\times$ speedup and $3,588,126\times$ energy

efficiency gain by mapping all of our inference operations to the stochastic domain. Thus, every operation is a simple bitwise computation that is easily implemented in PIM as well as parallelized. This is in contrast to the much more complex operations needed for a naive implementation of the Baseline-HD such as the element-wise operations.

C. Stochastic-HD vs State-of-the-Art Accelerators

Figure 4 compares the execution time and the energy consumption of Stochastic-HD with a CPU implementation of the Baseline-HD as well as a Naive PIM implementation. Naive-PIM simply maps the necessary operations for inference described in the state-of-the-art HD implementation in [7] onto the memory architecture as described in the state-of-the-art PIM work in [13]. However, as noted in Section I, simply implementing the Baseline-HD model into PIM leads to the element-wise multiplication and subsequent accumulation to bottleneck the parallelism and therefore performance.By applying SC in Stochastic-HD we are able to use simple bitwise operations throughout the inference process. With Stochastic-HD we achieve $52.8\times$ speedup and $378\times$ energy efficiency gain over Naive-PIM. However, Naive-PIM [7], [13] can achieve the same accuracy as the software implementation.

We also compared Stochastic-HD with two state-of-the-art HD-PIM implementations in [26] and SearcHD [1]. Stochastic-HD provides 13% higher accuracy on average than the [26] and 4.4% higher accuracy than SearcHD. Our results show that [26] is $6.1\times$ faster and SearcHD is $2.6\times$ faster than Stochastic-HD. However, Stochastic-HD can achieve $10{,}618\times$ energy efficiency over the design in [26] and $43.1\times$ energy efficiency over SearcHD. The higher energy efficiency of Stochastic-HD is due to the simpler bitwise stochastic operations. As compared to the best PIM design for HD [1], Stochastic-HD is 4.4% more accurate and $43.1\times$ more energy-efficient. Slightly worse performance of Stochastic-HD is the result of the Stochastic-HD's area-efficient approach as it uses the minimal memory required. Our design can however be extended to a bigger chip to achieve better performance. For example, a Stochastic-HD chip with $14\times$ ($174\times$) larger area provides $11\times$ ($134\times$) faster inference than the area-efficient Stochastic-HD, while consuming similar energy. The designs in [26], [1] are limited to the parallelism provided by HD computing and incur the latency of multi-bit computations. This means bigger Stochastic-HD chips are both faster and more accurate than the existing designs [26], [1].

VI. Conclusion

In this paper, we propose, Stochastic-HD, which combines HD Computing and SC to perform classification tasks in PIM with highly parallel operations. We use SC because a naive implementation of existing HD work in digital PIM results in a bottleneck when we need element-wise multiplications and subsequent accumulations for dot product. With Stochastic-HD during training and inference, we represent each element of the query hypervectors with bitstreams, and replace the bottleneck operations with simple bitwise operations. With Stochastic-HD, we were able to reach a comparable accuracy with the Baseline-HD. As compared to the best PIM design for HD [1], Stochastic-HD is also 4.4% more accurate and $43.1\times$ more energy-efficient.

Acknowledgements

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, in part by SRC-Global Research Collaboration grant Task No. 2988.001, and also NSF grants 1527034, 1730158, 1826967, 1830331, 1911095, and 2003277.

References

[1] M. Imani, X. Yin, J. Messerly, S. Gupta, M. Niemier, X. S. Hu, and T. Rosing, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[2] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.

[3] 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.

[4] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2015.

[5] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *ICRC*, pp. 1–6, IEEE, 2017.

[6] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, ACM, 2016.

[7] 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 *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[8] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*, pp. 37–172, Springer, 1969.

[9] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2016.

[10] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," in *DATE*, IEEE/ACM, 2019.

[11] 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*, 2019.

[12] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, pp. 53–62, ACM, 2019.

[13] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," in *IEEE/ACM ICCAD*, pp. 1–7, IEEE, 2018.

[14] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, pp. 1–19, 2013.

[15] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE transactions on computers*, vol. 60, no. 1, pp. 93–105, 2010.

[16] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "Vlsi implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688–2699, 2017.

[17] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, "Sign-magnitude sc: Getting 10x accuracy for free in stochastic computing for deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[18] M. Imani, S. Gupta, A. Arredondo, and T. Rosing, "Efficient query processing in crossbar memory," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*, pp. 1–6, IEEE, 2017.

[19] S. Kvatinsky *et al.*, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[20] "Uci machine learning repository: Isolet data set." http://archive.ics.uci.edu/ml/datasets/ISOLET.

[21] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[22] "Uci learning repository:daily and sports activities data set." https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities.

[23] "Uci machine learning repository: Pamap2 physical activity monitoring data set." https://archive.ics.uci.edu/ml/datasets/pamap2+physical+activity+monitoring.

[24] S. Benatti, E. Farella, E. Gruppioni, and L. Benini, "Analysis of robust implementation of an emg pattern recognition based control.," in *BIOSIGNALS*, pp. 45–54, 2014.

[25] "Uci machine learning repository: Cardiotocography data set." https://archive.ics.uci.edu/ml/datasets/cardiotocography.

[26] S. Datta, R. A. Antonio, A. R. Ison, and J. M. Rabaey, "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, pp. 439–452, 2019.