HyperSpike: HyperDimensional Computing for More Efficient and Robust Spiking Neural Networks

Justin Morris*†, Hin Wai Lui‡, Kenneth Stewart‡, Behnam Khaleghi*, Anthony Thomas*, Thiago Marback*,
Baris Aksanli†, Emre Neftci‡, and Tajana Rosing*
*University of California San Diego, La Jolla, CA 92093, USA
†San Diego State University, San Diego, CA 92182, USA
‡University of California Irvine, Irvine, CA 92697, USA
justinmorris@ucsd.edu, {hwluiu, kennetms}@uci.edu, {bkhalegh, ahthomas, tmarback}@ucd.edu,
baksanli@sdsu.edu, eneftci@uci.edu, tajana@ucsd.edu

***Abstract*—Today's Machine Learning(ML) systems, especially those running in server farms running workloads such as Deep Neural Networks, which require billions of parameters and many hours to train a model, consume a significant amount of energy. To combat this, researchers have been focusing on new emerging neuromorphic computing models. Two of those models are Hyperdimensional Computing (HDC) and Spiking Neural Networks (SNNs), both with their own benefits. HDC has various desirable properties that other Machine Learning (ML) algorithms lack such as: robustness to noise in the system, simple operations, and high parallelism. SNNs are able to process event based signal data in an efficient manner. In this paper, we create HyperSpike, which utilizes a single, randomly initialized and untrained SNN layer as feature extractor connected to a trained HDC classifier. HDC is used to enable more efficient classification as well as provide robustness to errors. We experimentally show that HyperSpike is on average $31.5\times$ more robust to errors than traditional SNNs. We also implement HyperSpike in hardware, and show that it is $10\times$ faster and $2.6\times$ more energy efficient over traditional SNN networks run on Intel's Loihi [1].**

I. Introduction

The Internet of Things (IoT) creates an ever increasing demand for new complex applications on low power edge devices. The slowdown of Moore's Law has pushed the current processing systems to their limits. Running data intensive workloads with large datasets on traditional cores results in high energy consumption and slow processing speeds. Most IoT applications today run on power constrained embedded systems. However, many such applications require the use of state-of-the art machine learning algorithms that consume a lot of power. This leads many systems to relay on cloud computing, causing user privacy issues when sending user data to be processed in the cloud rather than on their local device, and increasing the energy consumption of servers. With the growing importance of achieving energy efficiency and data privacy for IoT applications, there is a need to explore emerging low power neuromorphic computing models.

Neuromorphic computing platforms offer a promising efficient, low power alternative to conventional Von Neumann architectures for state-of-the-art ML algorithms, making them well-suited for complex yet power-constrained IoT and mobile applications. Neuromorphic systems take inspiration from the brain's event-driven dynamics, distributed architecture, in-memory computation, and massive parallelism to achieve such energy efficiency. Two such promising models are Hyperdimensional Computing (HDC) and Spiking Neural Networks (SNNs).

Brain-inspired HDC is a computation paradigm which represents data in terms of extremely large vectors, called *hypervectors* (HV). These HVs may have tens of thousands of dimensions and present data in the form of a pattern of signals instead of numbers [2]. By representing data with HVs, HDC reduces the complexity of operations required to process data. As a result, HDC uses extremely simple and highly parallel operations [3], and is very robust to errors [4]. Prior work has shown the suitability of HDC for various applications like activity recognition, face detection, language recognition, image classification, etc [5], [6].

Spiking Neural Networks (SNNs) simulate the dynamics and spike-based communication of biological neurons and are compatible with neuromorphic hardware platforms for online on-chip learning and inference [7]. The formalization of SNNs in terms of neural network building blocks reveals that SNNs are a type of Recurrent Neural Network (RNN) with internal states akin to the long short-term memory (LSTM), but in SNNs the time step is on the order of the modeled neural and synaptic processes. This implies that suitable applications for SNNs are those requiring memory and the modeling of temporal dynamics [8]. Furthermore, their dynamics and spike-based communication result in a rich temporal resolution, which make them particularly interesting for processing and learning spatiotemporal patterns recorded by neuromorphic sensors such as Dynamic Vision Sensors (DVS) [9]. However, inherent noise in emerging neuromorphic sensors, the variability in mixed-signal hardware that realize SNN operations, and limited weight precision all pose challenges when deploying systems for real-world applications [9]. Adding HDC for increased robustness could resolve these issues.

In this paper, we go beyond a simple combination of these two neuromorphic models, SNNs and HDC, to create HyperSpike. We show that in fact we do not need more than a single layer of the SNN, with randomized weights, as a feature extractor for the HDC layer. This removes the need to train SNN, and provides all the simplicity of HDC in terms of simple and fast training, and highly robust inference. While the success of the randomly initialized SNN with an HDC layer may sound surprising, there is precedent in the broader literature [10] [11]. In fact, in our work we show that

HyperSpike, with randomized first layer, is $31.5\times$ more robust to errors than SNNs, with comparable accuracy. Furthermore, we show that our hardware implementation of HyperSpike is $10\times$ faster and $2.6\times$ more energy efficient over the traditional SNN network running on Intel's Loihi [1].

II. Background and Related Work

A. Hyperdimensional Computing

Without loss of generality, we explain the steps HDC uses for classification, though other algorithms, e.g., clustering, follow the same procedure as well.

(1) Encoding: Let us assume a feature vector $\mathbf{F} = \{f_1,\ f_2, \ldots,\ f_n\}$, with n features in the original domain. The goal of encoding is to map this feature vector to a D dimensional space vector: $\mathbf{H} = \{h_1,\ h_2, \ldots,\ h_D\}$. The encoding first randomly generates D dense bipolar vectors with the same dimensionality as the original domain, $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_D\}$, where $\mathbf{p}_i \in \{-1, 1\}^n$. The inner product of a feature vector with each generated vector gives us a single dimension of a hypervector in high-dimensional space. To encode a feature vector into a hypervector, we perform a matrix vector multiplication between the projection matrix and the feature vector using: H = PF.

(2) Training: The simplicity of HD training makes it distinguished from conventional learning algorithms. Consider hypervector $\mathcal{H}_i$ as the encoded hypervector of input i with the procedure explained above. Each input i belongs to a class j, so we further annotate $\mathcal{H}_i^j$ to show the class j of input i, as well. HD training simply adds all hypervectors of the same class to generate the final model hypervector. Therefore, the class hypervector of label j, denoted by $\mathcal{C}^j$, is:

$$\mathcal{C}^j = \mathcal{H}_0^j + \mathcal{H}_1^j + \cdots = \sum_i \mathcal{H}_i^j. \tag{1}$$

We simply accumulate the encoded hypervectors for which their original input belongs to class j. The class HVs are then binarized by taking the sign of the final accumulated class HVs.

(3) Similarity checking: The inference searches for the most similar class hypervector to the encoded query. When hypervectors are binary, such as in this work, the search is done using Hamming distance. When hypervectors are not binary, cosine similarity is used.

B. Spiking Neural Networks

SNNs can be formulated as a type of recurrent neural network with binary activation functions [8]. With this formulation, SNN training can be carried out using standard tools of autodifferentiation. In particular, to best match the dynamics of existing digital neuromorphic hardware implementing SNNs [1], our neuron model consists of a discretized Leaky Integrate and Fire (LIF) neuron model with time step Δt written in the form of a Spike Response Model [12]:

$$\begin{aligned} P_j^{t+\Delta t} &= \alpha P_j^t + S_{in}^t, \\ R_i^{t+\Delta t} &= \alpha R_i^t + \alpha U_i^t S_i^t, \\ U_i^{t+\Delta t} &= \sum_j W_{ij} P_j^{t+\Delta t} - R_i^{t+\Delta t}, \\ S_i^{t+\Delta t} &= \Theta(U_i^{t+\Delta t}). \end{aligned} \tag{2}$$

where the constant $\alpha = \exp(-\frac{\Delta t}{\tau_{\text{mem}}})$ reflects the decay dynamics of the membrane potential during a Δt timestep, where τ_{mem} and is the membrane time constant. The time step in our experiments was fixed to $\Delta t = 1$ms. R_i here represents the reset and refractory period of the neuron, and state P_i is the pre-synaptic trace that captures the leaky dynamics of the membrane potential. $S_i^t = \Theta(U_i^t)$ represents the spiking non-linearity, computed using the unit step function, where $\Theta(U_i) = 0$ if U_i is smaller than threshold U_{th}, otherwise 1. We distinguish here the input spike train S_{in}^t from the output spike train S^t. For the purposes of computing the gradient, the derivative of Θ is replaced with the derivative of a smooth function, the fast sigmoid function [13]. This Follows the surrogate gradient approach [8].

C. Related Work

HDC is light-weight enough to run with acceptable performance on CPUs [14]. However, utilizing a parallel architecture can significantly speed up HDC execution time. Imani *et al.* showed two orders of magnitude speed up when HD runs on GPU [15]. Salamat *et al.* proposed a framework that facilitates fast implementation of HDC algorithms on an FPGA [16]. However, to get the best performance out of HDC ASIC accelerators should be used [3]. HDC is able to achieve the best performance in ASIC due to the bit-level operations in HDC. Furthermore, there have been multiple works on implementing HDC on new emerging computing hardware such as ReRAM crossbars [17], [18]. However, these works do not evaluate the hardware level errors that these architectures incur. In this paper, we empirically evaluate the robustness of HyperSpike to these hardware level errors by simulating various levels of bit error rates in software.

Several works show that HDC is inherently robust to noise [19], [20], [4]. Work in [19] investigated the robustness of HDC to RTL errors and found that HDC-based approach tolerated $8.8\times$ higher probability of bit-level errors, similar to [20]. Work in [4] showed that HDC is also robust to wireless communication errors.

The semantic pointer architecture [21] implements a type of HD-computing using SNNs and the Neural Engineering Framework. Since then other approaches to using semantic pointer architectures for SNN computation have been proposed such as representing phasors as spike times [22] and representing hypervectors as Sparse Block-Codes [23]. However no implementation of this and similar methods in neuromorphic hardware have been reported to date [24].

HyperSpike differs from previous approaches because it combines two separate blocks, namely SNNs and HD computing separately, rather than *implementing* SNNs using HD or vice versa. We used the Intel Loihi [1] for evaluating the neuromorphic processing component of HyperSpike. The Intel

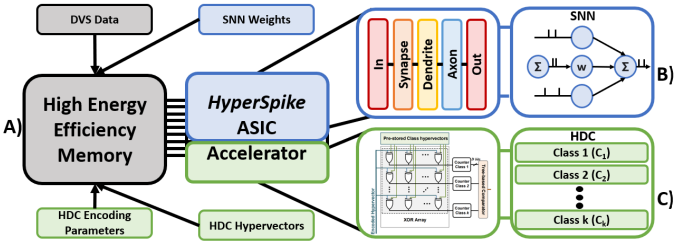


Fig. 1. Overall System Architecture of HyperSpike.

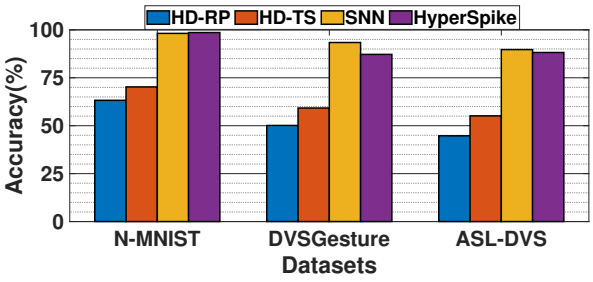


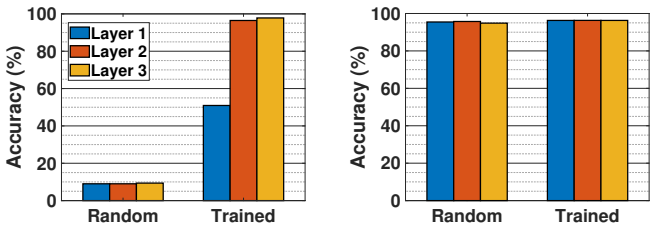
Fig. 2. Comparison of Using Existing HDC vs Traditional SNNs

Loihi is a 60-mm^2 neuromorphic processor fabricated in Intel's 14-nm process that integrates a wide range of features such as hierarchical connectivity, dendritic compartments, synaptic delays, and programmable synaptic learning rules [1]. Results from previous studies have demonstrated that brain-inspired networks implemented on the Intel Loihi, such as SNNs using precise spike-timing relationships from event-based data processing, perform certain computation with orders of magnitude lower latency and energy compared to conventional state of the art approaches such as those based on feedforward deep neural networks [25]. Additionally, the Loihi is capable of online learning allowing for low power edge learning without needing a cloud [7]. It uses SRAM for state, causing the state bit cost to be high compared to conventional processors that use DRAM. To reduce the cost of the Loihi chip one could store state in ReRAM, but at the cost of high bit error rates [26]. We evaluate the impact of utilizing emerging computing hardware with errors and show how HyperSpike combats them.

III. HyperSpike: Combining SNNs and HDC

HyperSpike starts with (A), a high energy efficiency memory where HyperSpike stores the parameters for the SNN and HDC layers. HyperSpike is accelerated by an ASIC chip that utilizes 56 Intel Loihi cores for acceleration of the SNN layer shown in (B). The output of the SNN layer at the final time step of the sequence is connected to (C), the inference accelerator for HDC based on tinyHD [3]. This accelerator first encodes the feature vector from the output of the SNN as shown in Section II. Then, the encoded hypervector (HV) is compared to the pre-trained class HVs using Hamming distance metric.

HDC Challenges: Figure 2 shows the accuracy of HDC on three DVS datasets. We tested two encoding methods: one that targets feature vectors using random projection (HD-RP) [6] and is also the encoding we use for the HDC layer of HyperSpike, as well as HDC encoding for times series



(a) SNN (b) HyperSpike

Fig. 3. Impact of Random Weights on HyperSpike and Traditional SNNs Across Different Layers of the SNN using the NMNIST dataset.

data (HD-TS) [27]. With an average HDC accuracy of 57%, compared to an average accuracy of 94% on SNNs, we can see that existing HDC systems are not able to classify DVS data as accurately. To combat this, we use randomly initialized first layer of an SNN to preprocess the DVS data. This transforms the event based data from the neuromorphic sensors into more representational feature vectors that HDC can better leverage.

In Memory Errors: When a query comes in for processing on HyperSpike, the data is first stored in memory, or (A) in Figure 1. This memory comes at the cost of introducing bit errors in the data. Previous designs with emerging computing hardware assumed that the technology had ideal characteristics and no errors. However, various emerging computing technologies, especially those that offer significantly more energy efficiency often incur bit level errors on the data they store. We simulate these errors in Section IV. These errors extend to all data stored in memory, such as SNN weights and hypervectors for HDC. However, as we describe below, HDC is able to overcome these errors due to its robustness. Although we model our memory separately from our compute chip, our experiments can be generalized to other applications where bit error rates occur on the model parameters. For instance, our results would extend to implementing HyperSpike entirely on ReRAM storing our model parameters and computing on the same ReRAM arrays. This can be further generalized beyond just ReRAM to other hardware with bit error rates (BER). Such as, emerging Non-volatile memories, low voltage memories, or even wireless communication [26].

SNN Acceleration of HyperSpike: After the data is in memory, it is then sent for processing in the custom neuromorphic processor that realizes the dynamics of the SNN layer of HyperSpike. The acceleration of the SNN layer is handled by 56 Loihi cores. For one-to-one mapping of the trained network in Loihi, the SNN is modeled with a functional Loihi simulator and the normalized post-synaptic response is used for temporal error credit assignment during gradient-based learning. Since Loihi only supports integer weights, a strategy of full precision shadow weights is used, which are quantized during the forward inference phase only [7].

HyperSpike Removes SNN Training: Due to the memory implemented by the accelerator for HyperSpike being subject to errors, we compare SNN models with trained weights to models using randomly generated weights. Figure 3 compares the accuracy results of using random weights and trained weights with both HDC as the classifier and a more traditional MLP as the classifier on N-MNIST. The results show that

TABLE I
ACCURACY OF HyperSpike VS OTHER SNN CLASSIFIERS AT FULL PRECISION AND QUANTIZED TO 16 BITS. PERFORMANCE NUMBERS ARE RELATIVE TO QUANTIZED SNN+MLP

Dataset	SNN+MLP [8]	Quantized SNN+MLP [28]	SNN+VAE [29]	Quantized SNN+VAE [29]	HyperSpike	Quantized HyperSpike		
Metrics	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy	Speedup	Energy Efficiency
N-MNIST	98.4%	97.4%	98.2%	97.4%	98.6%	95.2%	1.1×	1.1×
DVSGesture	93.8%	86.8%	83.6%	72.8%	87.2%	85.3%	14×	2.2×
ASL-DVS	89.7%	87.1%	89.7%	87.1%	88.2%	87.8%	14.9×	4.6×

when using a traditional classifier, such as an MLP, training of the SNN weights is necessary. In contrast, HDC is able to achieve high accuracy even with random SNN weights. Not training the weights does come at a slight accuracy drop, but HyperSpike gains the benefit of not needing to perform any costly SNN training. We got similar results for all datasets.

While the success of the randomly initialized SNN with an HDC layer may sound surprising, there is precedent in the broader literature on deep learning for the success of random networks. Seminal work from [10] and reservoir computing showed that a randomly initialized shallow multilayer perceptron can learn to approximate any continuous function just by training the final layer – similar to how we here train only the final HD layer. More recently, these properties have been extended to more general types of neural networks with random initializations [11]. A key implication of this work is that weights in the early layers of networks generally move very little over the training process, suggesting that a random initialization is sufficient to obtain good performance while only training the last few layers. Moreover, randomly initialized neural networks are able to store and recall information like sequences of pixel values [30]. These random architectures have previously been used successfully to classify simple images of handwritten characters Work in [30] showed that this property extends to randomly initialized recurrent neural networks which are functionally similar to SNNs. In summary, there is a broad literature showing that even randomly initialized neural networks possess strong properties for learning, which lend credence to our findings here. However, our findings show that this random initialization of SNNs only works when using HDC as the classifier.

HyperSpike Only Needs One SNN Layer: Figure 3 also shows the accuracy of attaching HDC to different layers of the SNN on N-MNIST. The data in Figure 3b shows that HDC attached to SNN layers is able to achieve high accuracy across all layers of the attached SNN, unlike a standalone trained SNN shown in Figure 3a that only achieves high accuracy in the last layer. In fact, we can see that HDC achieves comparable accuracy to it's maximum accuracy with just one SNN layer. This trend is true across all the datasets we tested. Therefore, HyperSpike utilizes just one SNN layer to act as a preprocessing step to transform the data from DVS data to a feature vector. This allows HyperSpike to save even more energy over traditional SNN networks.

HDC Acceleration for HyperSpike: After the output of the SNN is sent to the HDC module shown in Figure 1(C), the data is first encoded into HD space. This is done with the random projection encoding described in Section II. This transforms the data from the lower dimensional feature vector into a $D = 10,000$ hypervector. This gives HDC its robustness as the information is spread out over a $10,000$ dimensional vector. We'll call this the query HV. Once HyperSpike has the query HV, it is then compared to the pre-trained class HVs, which are trained offline and stored in memory (A). Since all of the HVs use binary numbers, HyperSpike can use Hamming distance to calculate the similarity of the query HV with all the class HVs. This is done with simple XOR blocks to count the number of mismatches. The class with the least number of mismatches is chosen as the output.

IV. EVALUATION

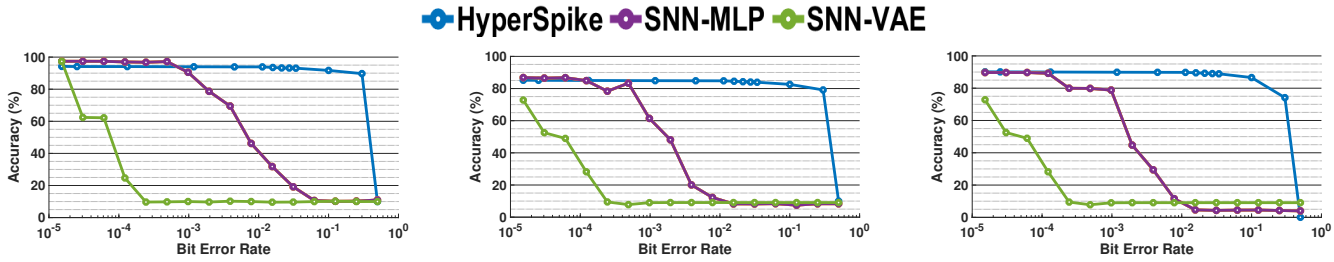
A. Experimental Setup

We verified the functionality of HyperSpike using both software and hardware implementations. In software, we implemented HyperSpike on an Intel Core i7 7600 CPU using an optimized C++ implementation. We additionally used C++ to simulate bit error rates in HyperSpike. For SNN experiments we estimate the performance of the 56 Loihi cores in the accelerator for HyperSpike by using Intel's Loihi the Intel Nahuku board [1]. For HDC acceleration our design is based on [3]. We utilized SystemVerilog at the RTL level and used Synopsys Design Compiler with the 45 nm open source NanGate cell library to synthesize [31]. For a good comparison with [1], we scale the results to 14 nm by using 14 nm technology in CACTI for the memories area and power, and adopting the scaling trend of Intel [32] for the logic cells area. To scale the power, we first obtained the 45 nm power consumption reported by Synopsys Power Compiler, then, we obtained the scaling trend using HSPICE simulations with Predictive Technology Model (PTM) [33] models. For comparison, we compare with SNNs using MLP as the classifier as well as VAE as the classifier.

We tested our proposed approach on 3 popular DVS applications: Handwritten digit Recognition (N-MNIST) [34], Gesture Recognition (DVSGesture) [35], and Sign Language Recognition (ASL-DVS) [36].

B. Accuracy of HyperSpike vs SoA in the Presence of BER

Table I compares HyperSpike with SNNs using different classifiers at the output of the SNN network. The results show that HyperSpike with a full precision SNN is able to achieve similar accuracy to State-of-the-Art solutions. HyperSpike is able to achieve accuracy within $2.6\%(1\%)$ of the other classifiers (quantized). Although HyperSpike loses some accuracy when compared to State-of-the-Art in some cases, HyperSpike is able to achieve other significant benefits from using HDC.



(a) N-MNIST (b) DVSGesture (c) ASL-DVS

Fig. 4. Impact of Varying Levels of Bit Error Rates on the Accuracy of Quantized HyperSpike vs Other Quantized SNN Models.

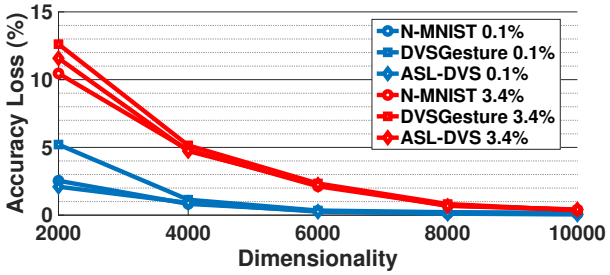


Fig. 5. Impact of Dimensionality on the Robustness of HyperSpike

For instance, HDC makes HyperSpike significantly more energy efficient as HyperSpike only needs to use the first layer of the SNN as well as HDC having simple parallel operations easily accelerated in ASIC hardware. HyperSpike additionally gains significantly more robust capabilities over the current State-of-the-Art SNN classifiers.

Figure 4 shows a comparison of HyperSpike with SNNs using different classifiers at the output of the SNN network. All SNN networks are quantized to 16 bit fixed-point precision [28]. This is because when there are bit flips in a system, having a fixed-point data representation reduces the impact of the bit flips. For instance, if an exponent bit is flipped in a floating point number, the resulting numerical error is more significant than flipping a mantissa bit. Taking this further, binary representations offer the highest resilience to bit flip errors as no matter which bit is flipped, the resulting numerical error is the same. This gives HyperSpike an advantage over SNN classifiers as the HDC classification portion of HyperSpike uses binary quantization. We define empirically how much more robust one classifier is than the other by the ratio of accuracy lost when there are no bit flip errors. Our experiments show that HyperSpike is $31.4\times$ more robust than SNNs using MLPs (lost $31.4\times$ less accuracy than SNNs using MLPs) as a classifier when the BER is 0.1%, which is a typical error rate for systems that have high bit errors. HyperSpike is $58.3\times$ more robust than SNNs using MLPs as a classifier when the BER is 3.4%, or at worst case scenario BERs. We additionally tested SNNs using a VAE for classification and the results show that similar to MLPs, VAEs do not offer as much robustness as HDC.

Although we model errors to occur in a separate chip for memory, our experiments can be generalized to other applications where BERs occur on the same model parameters. Our results would extend to implementing HyperSpike entirely on a ReRAM architecture such as [18]. They can similarly also extend to wireless communication errors if the parameters are sent and shared across different devices in a network. This indicates that one could create a compute architecture for HyperSpike that utilizes BER emerging hardware and not need to add overhead of error correction as HyperSpike is robust to the errors due to its HDC layer.

Figure 5 tests HyperSpike with varying dimensionalities and two different error rates. The lines in red represent worst case BER environments and the lines in blue represent typical high BER environments. The results indicate that HyperSpike robustness scales with the dimensionality of the HDC model used for classification. As we increase the dimensionality of the HDC model, the accuracy loss decreases. This is consistant with prior work, which has shown that HDCs robustness is a function of high dimensionality [4].

C. HyperSpike *Accelerator Performance and Statistics*

Figure 6 compares HyperSpike with a more traditional SNN using an MLP as the output classifier running on an Intel Loihi chips for acceleration. Our accelerator for HyperSpike consists of a custom neuromorphic processor for SNN, but we are evaluating using Loihi to get an estimate for the SNN layer and [3] for the HDC layer. The results show that HyperSpike is $2.6\times$ more energy efficient and achieves a speedup up of $10\times$ over traditional SNNs. There are two main reasons for this. (1) HyperSpike only uses one SNN layer. This is significant because the SNN portion of the process takes a large portion of time and energy for inference. Therefore, cutting costs at the SNN layers leads to significant improvements. (2) Our HDC layer is significantly more energy efficient and faster than the MLP layer of a traditional SNN.

We can see that the difference in performance in the execution time and energy on the N-MNIST dataset is minimal because the N-MNIST network uses only two fully connected layers. Therefore, because most of the performance gain from HyperSpike is from shrinking the SNN, the smaller the original network, the less performance gain HyperSpike provides. On the other hand, the other two datasets require larger networks and HyperSpike provides significantly better performance. Additionally, we see the largest difference in execution time on the ASL-DVS dataset because the network is spread across the greatest number of cores incurring communication overhead across them.

Another advantage of our accelerator over State-of-the-Art is reduced chip area. Figure 6 compares the area needed to

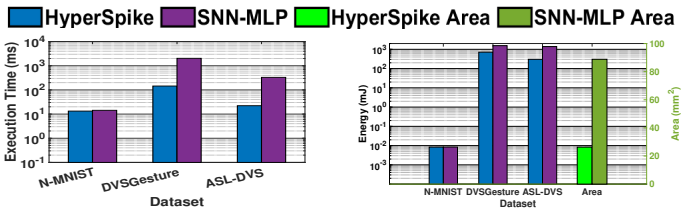


Fig. 6. Comparison of HyperSpike vs Traditional SNN with Intel's Loihi on Energy Efficiency, Execution Time, and Chip Area.

accelerate HyperSpike with the area needed to accelerate a traditional SNN with an Intel Loihi chip. For the traditional SNN, we only include the percent area of the Loihi chip that is actually used for acceleration. For HyperSpike, we add up the area from the number of Loihi cores our chip needs plus the area needed for our HDC ASIC. For both HyperSpike and SNN-MLP we compare the chip area needed for the largest dataset, DVSGesture. As a result, this hardware could run all three datasets. The data shows that our accelerator is $3.4\times$ smaller than a traditional SNN accelerator.

V. Conclusion

In this paper we go beyond a simple combination of two neuromorphic models, SNNs and HDC, to create HyperSpike. The first layer of HyperSpike is a randomly initialized SNN layer that does not need to be trained. This layer processes the event based signal data from a neuromorphic sensor and outputs feature vectors. Then, the trained HDC layer interprets these feature vectors to perform classification. By combining SNNs and HDC in this way, HyperSpike is able to achieve high classification accuracy with HDC on DVS data, while being much smaller, faster and more energy efficient. Our results show that HyperSpike is $31.4\times$ more robust to errors than traditional SNNs. Our HW implementation of HyperSpike is $10\times$ faster and $2.6\times$ more energy efficient over traditional SNN networks running on Intel's Loihi [1].

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, 1652159.

References

[1] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, P. Joshi, A. Lines, A. Wild, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. PP, no. 99, pp. 1–1, 2018.

[2] 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.

[3] B. Khaleghi, H. Xu, J. Morris, and T. Š. Rosing, "tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 408–413, IEEE, 2021.

[4] J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, and T. Rosing, "Hydrea: Towards more robust and efficient machine learning systems with hyperdimensional computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 723–728, IEEE, 2021.

[5] 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.

[6] M. Imani *et al.*, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[7] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci, "Online few-shot gesture learning on a neuromorphic processor," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, pp. 512–521, Oct 2020.

[8] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.

[9] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, *et al.*, "Event-based vision: A survey," *arXiv preprint arXiv:1904.08405*, 2019.

[10] A. Rahimi and B. Recht, "Uniform approximation of functions with random bases," in *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pp. 555–561, IEEE, 2008.

[11] B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari, "Linearized two-layers neural networks in high dimension," *The Annals of Statistics*, vol. 49, no. 2, pp. 1029–1054, 2021.

[12] W. Gerstner and W. Kistler, *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[13] F. Zenke and S. Ganguli, "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks," *Neural Computation*, vol. 30, pp. 1514–1541, 06 2018.

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

[15] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

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

[17] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2018.

[18] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 802–815, IEEE, 2019.

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

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

[21] C. Eliasmith, T. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[22] E. P. Frady and F. T. Sommer, "Robust computation with rhythmic spike patterns," *Proceedings of the National Academy of Sciences*, vol. 116, no. 36, pp. 18050–18059, 2019.

[23] E. P. Frady, D. Kleyko, and F. T. Sommer, "Variable binding for sparse distributed representations: Theory and applications," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2021.

[24] 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 nanoscale hardware," 2021.

[25] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[26] M. Davies, "Lessons from loihi: Progress in neuromorphic computing," in *2021 Symposium on VLSI Circuits*, pp. 1–2, 2021.

[27] A. Rahimi, A. Tchouprina, P. Kanerva, J. d. R. Millán, and J. M. Rabaey, "Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials," *Mobile Networks and Applications*, pp. 1–12, 2017.

[28] H. W. Lui and E. Neftci, "Hessian aware quantization of spiking neural networks," 2021.

[29] K. Stewart, A. Danielescu, L. Supic, T. Shea, and E. Neftci, "Gesture similarity analysis on event data using a hybrid guided variational auto encoder," *arXiv preprint arXiv:2104.00165*, 2021.

[30] E. P. Frady, D. Kleyko, and F. T. Sommer, "A theory of sequence indexing and working memory in recurrent neural networks," *Neural Computation*, vol. 30, no. 6, pp. 1449–1513, 2018.

[31] "Nangate open cell library." https://si2.org/open-cell-library/.

[32] M. T. Bohr and I. A. Young, "Cmos scaling trends and beyond," *IEEE Micro*, vol. 37, no. 6, pp. 20–29, 2017.

[33] "Predictive technology model." http://ptm.asu.edu/.

[34] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, nov 2015.

[35] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7243–7252, 2017.

[36] Y. Bi, A. Chadha, A. Abbas, , E. Bourtsoulatze, and Y. Andreopoulos, "Graph-based object classification for neuromorphic vision sensing," in *2019 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2019.