Integrating Event-based Dynamic Vision Sensors with Sparse Hyperdimensional Computing: A Low-power Accelerator with Online Learning Capability

Michael Hersche*★, Edoardo Mello Rella*★, Alfio Di Mauro★, Luca Benini†★, Abbas Rahimi‡★
★Integrated Systems Laboratory, D-ITET, ETH Zurich, Switzerland
† DEI Department, University of Bologna, Italy
‡EECS Department, UC Berkeley, USA
Emails: edoardom@student.ethz.ch, {hersche, dimauro, benini, abbas}@iis.ee.ethz.ch

Abstract

We propose to embed features extracted from event-driven dynamic vision sensors to binary sparse representations in hyperdimensional (HD) space for regression. This embedding compresses events generated across 346×260 differential pixels to a sparse 8160-bit vector by applying random activation functions. The sparse representation not only simplifies inference, but also enables online learning with the same memory footprint. Specifically, it allows efficient updates by retaining binary vector components over the course of online learning that cannot be otherwise achieved with dense representations demanding multibit vector components. We demonstrate online learning capability: using estimates and confidences of an initial model trained with only 25% of data, our method continuously updates the model for the remaining 75% of data, resulting in a close match with accuracy obtained with an *oracle* model on ground truth labels. When mapped on an 8-core accelerator, our method also achieves lower error, latency, and energy compared to other sparse/dense alternatives. Furthermore, it is 9.84× more energy-efficient and 6.25× faster than an optimized 9-layer perceptron with comparable accuracy.

CCS Concepts

• **Computer systems organization → Embedded software**; • **Computing methodologies → Machine learning**.

Keywords

Sparse HD computing, robotics, event-based cameras, regression.

1 Introduction

Neuromorphic dynamic vision sensors (DVSs) asynchronously report intensity changes in a scene by generating a set of sparse events [2, 21]. Hence, the output of a DVS is not a sequence of images but a stream of asynchronous events. This property of seeing sparse events in time as opposed to full image pixels at a fixed frame rate allows DVS to capture motion better than a classical camera. DVSs further provide high dynamic range, high temporal resolution, very low latency, and reduce redundant data for downstream processing that make them useful in low-power applications such as robotics, active perception, self-driving cars, real-time tracking, and gesture recognition [1, 3, 11, 13, 14, 20].

In a bio-inspired DVS, each pixel acts as an *independent* circuit that outputs the light intensity changes, as events, at the time they occur. These asynchronous events can be naturally processed by spiking neuromorphic processors such as TrueNorth [1], or Loihi [20]. However, it is not clearly evident how to combine DVS with other non-spiking machine learning approaches that favor efficient realization using digital, synchronous accelerators. This calls for a computing paradigm that can effectively transform the asynchronous sparse events—generated across thousands of sensing elements—to a compressed, temporal, and *i.i.d.* representation that can be readily used for learning and inference in a low-power digital accelerator; the resulting representations should also enable direct interaction with other cognitive modules such as motor control, planning, etc [4, 14].

One viable option, inspired by the very size of the brain, is to use hyperdimensional (HD) representations (e.g., with 10,000 i.i.d. components [8]) that enable efficient digital processing at low signal-to-noise ratio [17, 19] as well as analog in-memory computing [9]. HD representation also serves as a common language for direct coupling sensory and motor systems. In an initial effort, perceptions and actions are encoded in a single space using dense binary HD vectors [14]. More specifically, DVS events are bound with a drone's velocities to enable active perception in an HD space that is semantically informed and meaningful by construction. Such HD representations find a variety of use cases in different robotic applications (see [15] for an overview). We focus on sparse binary HD representations [16] that easily support online learning, and allow natural interaction with a sparse distributed memory (SDM) as long-term memory [7]. The interaction with SDM already enabled interesting features such as sequence learning and retrieval [12], and scalability to a large number of sensory inputs in wearable robots [22].

In this paper, we propose to embed features extracted from DVS events to sparse binary HD vectors using novel randomized activation functions encoding inspired by SDM (Section 3). This sparse encoding simplifies learning and inference procedures, and therefore allows the model to be updated during inference without any additional memory requirement (Section 4). Experimental results, in Section 5, on MVSEC [24] dataset for regression tasks demonstrate that: 1) Our sparse encoding achieves 7.7% lower average relative

*Both authors contributed equally to this research.

ISLPED '20, August 10–12, 2020, Boston, MA, USA

ACM ISBN 978-1-4503-7053-0/20/08…$15.00
https://doi.org/10.1145/3370748.3406560

pose error (ARPE) than state-of-the-art dense binary encoding [14]. 2) When trained initially with 25% of data, our method can provide estimates and confidences to continuously update the model for the remaining unseen data, achieving lowest ARPE, almost the same as an oracle updated with ground truth labels. The updated vectors remain in the binary space during the entire procedure of online model update. 3) Measurements on a commercial ultra-low power 8-core accelerator show that our method achieves lower latency, and energy compared to other dense/sparse alternatives. Compared to an optimal 9-layer perceptron, it also achieves lower latency (0.28 ms vs. 1.75 ms), and energy (5.0 μJ vs. 49.2 μJ) per inference, with only 2.1% higher ARPE (0.1608 vs. 0.1575). Our code is available open-source[1].

2 Background

2.1 Event-driven Dynamic Vision Sensors

In contrast to traditional frame-based cameras, neuromorphic dynamic vision sensors (DVSs) report only intensity changes per pixel, called events, which naturally respond to motion in the scene [11]. Instead of tracking the absolute intensity per pixel, DVS events only contain the information about the polarity of the change of the pixel. This results in a stream of sparse and asynchronous events, each being a tuple $e(x, y, t, p)$ with spatial coordinates x and y, time of event t with a resolution in μs, and p the polarity of the intensity change. DVSs offer many advantages compared to frame-based cameras: 1) higher dynamic range (140 dB instead of 60 dB); 2) low latency; 3) low power consumption; 4) lower bandwidth requirements, thanks to the sparsity of events [6].

The DVS stream can be used to estimate motion, depth, or optical flow using different regression techniques [6, 11, 14]. We typically distinguish between methods that operate on an event-by-event basis or on groups of events [6]. In the latter case, events $e(x_k, y_k, t_k, p_k)$ are usually projected on a 2-dimensional image plane by counting events over a certain period T:

$$\mathbf{I}(x, y) = \sum_{t_k \in T} \delta(x - x_k, y - y_k), \tag{1}$$

where $\delta(\cdot, \cdot)$ is the 2-dimensional Kronecker delta function and $\mathbf{I} \in \mathbb{N}^{n_x \times n_y}$ the pixel-wise histogram of events, which we call *event-count-image*. For example, $\mathbf{I}(2, 3) = 5$ means the event $e(2, 3, t, p)$ occurred 5 times within the period T. The dimension of the event-count-image n_x and n_y correspond to the spatial resolution of the DVS. The polarity of the events is ignored. This event-count-image comes with many advantages, such as ease of computation, robustness towards noise, or preservation of the motion information stored within the event stream [14]. The event count can be further refined with the *time-image* $\mathbf{T} \in \mathbb{R}^{n_x \times n_y}$ [13], which represents the average timestamp over period T of the events per pixel:

$$\mathbf{T}(x, y) = \frac{1}{\mathbf{I}(x, y)} \sum_{x_k = x, y_k = y} t_k. \tag{2}$$

Further, the local spatial *gradient-image* of $\mathbf{T}$ along the x- and y-axis is computed to get $\mathbf{G}_x$ and $\mathbf{G}_y$, respectively. The gradient images are used to determine six global descriptors [13] (see Fig. 1). These features include the sum of all gradients in $\mathbf{G}_x$ and $\mathbf{G}_y$, the sum of element-wise inverse of gradients in $\mathbf{G}_x$ and $\mathbf{G}_y$, the sum of gradients in both directions multiplied by their position relative to image center, and the sum of ratios between the ℓ_2-norm of gradients and the ℓ_2-distance between each pixel location and the image center. These features are used for velocity estimation [14], and motion compensation [13].

[1] https://github.com/iis-eth-zurich/hd_dvs

2.2 MVSEC dataset

We use the multivehicle stereo event camera (MVSEC) [24] dataset that features event recordings from two experimental mDAVIS-346B cameras with a resolution of 346×260 pixel ($n_x \times n_y$), mounted on a car together with IMUs and GPS sensors for determining the ground truth velocities. Therefore, this dataset serves for regression tasks estimating velocities based on DVS data. It consists of five outdoor test rides, two recorded during the day and three in the evening, with duration varying from 262 s to 653 s. The rides consist of multiple turns with a maximum angular velocity of 35.5 °/s.

2.3 Hyperdimensional Computing

To further process DVS events, hyperdimensional (HD) computing is a viable alternative. HD computing is inspired by the very size of the brain's circuit, where neural activity patterns are modeled as points, or *hypervectors*, in high-dimensional space [8]. Hypervectors are d-dimensional (the number of dimensions is in the thousands), *holographic*, and (pseudo)random with independent and identically distributed (i.i.d.) components. In the following, we simply denote them as *vectors*. HD computing starts with assigning *atomic* vectors to each symbol and storing it in an item-memory (IM). Dense atomic vectors are generated by drawing d elements of an i.i.d. binomial distribution with equal probability $p = 1/2$, i.e., the number of '0' and '1' is balanced. Distance between vectors is determined by Hamming distance. Due to its high dimensionality, atomic (random) vectors are orthogonal with very high probability.

HD computing comes with a set of well-defined vector operations on dense codes: 1) *binding* of two vectors using element-wise XOR; 2) *permutation* of a vector; 3) *bundling* a set of vectors using the element-wise majority operation. All these operations preserve dimensionality. Binding and permutation produce dissimilar vectors, whereas bundling preserves similarity, and thus is able to represent sets of vectors. When bundling too many vectors using binary majority operation, the recall performance is dropped due to the capacity limit [10]. As an alternative bundling operation, binary vectors are mapped to *bipolar* vectors ($\{-1, +1\}^d$) and accumulated element-wise using integer counters. This bundling produces HD vectors with integer components. The similarity measure transforms to the inner product between bundled integer vector and bipolar vector.

Principles of HD computing are also applied to design systems capable of solving classification tasks by 1) *symbolization*: mapping discrete or continuous input values to an atomic vector; 2) *encoding*: combine multiple vectors using binding, permutation, and bundling; 3) *classification*: find the best matching vector in associative memory (AM), which stores bundled sets for every class. It was successfully applied, e.g., in text analytics [19], several biosignal processing applications [18], robotics [15], etc [10, 17]. More recently, HD computing was used for regression to estimate velocities [14] based on DVS streams from the MVSEC dataset. Each value of six global features was first quantized to one of n levels and mapped to the corresponding $d/6$-dimensional vector using thermometer encoding. Finally, all six mapped vectors are concatenated to form a d-dimensional vector and classified using the AM with Hamming distance. The AM stores for every velocity a binary

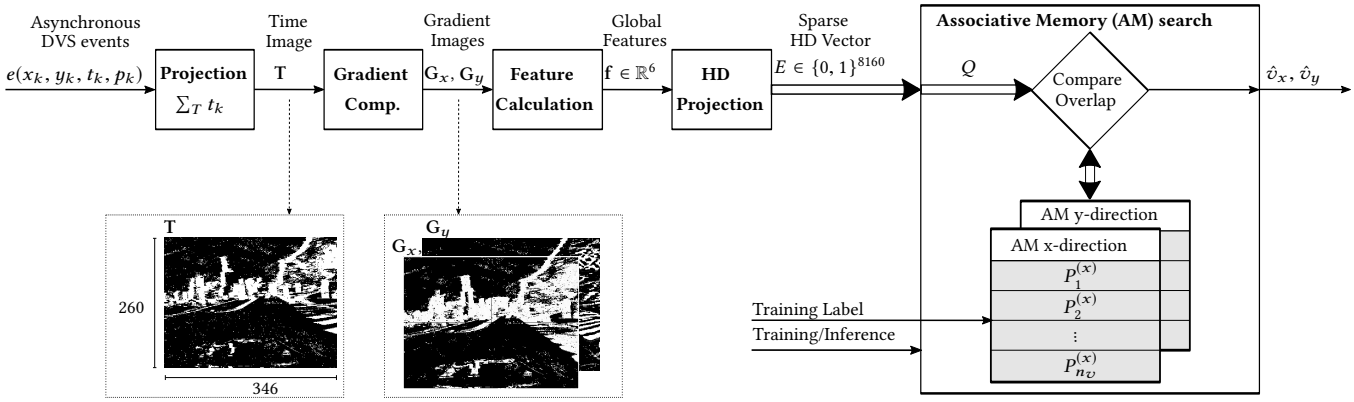

Figure 1: Training and inference pipeline for estimating velocities based on asynchronous DVS events: Events occurring within a time window of 0.05 s are projected to time-image T and transformed to six global features, represented by f. The HD projection block maps the features to a sparse binary 8160-dimensional vector. The AM searches for the best matching prototype in x- and y-direction and yields the final velocity estimations.

vector using the majority operation. This approach achieves similar prediction error compared with a more traditional vision approach using neural networks to predict velocities [14].

3 Learning Sparse HD Representations from DVS Events

In the following sections, we present the main contributions of the paper. We map the six global features from DVS events to *sparse* binary vectors (Section 3), which allows more efficient learning and inference compared to dense representations, and therefore enables online model update in the binary space (Section 4). Our proposed processing chain is illustrated in Fig. 1. First, asynchronous DVS events of a time window of 0.05 s are used to compute the time-image $\mathbf{T}$, which has 346×260 elements. After determining the local gradient images $\mathbf{G}_x$ and $\mathbf{G}_y$, the feature vector $\mathbf{f}$, containing the six global descriptors, is computed. The features are mapped to binary vectors of dimension d=8160, which can be efficiently represented by 255 32-bit values. The AM search is the final regression block, providing a training and inference mode. It stores n_v prototypes per x- and y-direction representing different velocity levels. In inference, the query vector Q is compared against all prototypes in both directions; the velocity level with the corresponding highest overlapping prototype is selected to be the estimated velocity.

3.1 Mapping Global Features to Sparse Atomic Vectors

This section describes how we map global features $\mathbf{f} \in \mathbb{R}^{n_f}$ to the d-dimensional sparse binary vectors $E \in \{0,1\}^d$.

3.1.1 Selective Kanerva Coding (SKC): A brain-inspired approach is to use Kanerva coding with a sparse distributed memory (SDM) [7]. The SDM represents $\mathbb{R}^{n_f}$ with d prototype vectors $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_d$ of dimension n_f, where the entries are drawn from an i.i.d. normal distribution. Given a query vector $\mathbf{f}$, SDM measures the ℓ_2-distance to all prototypes and activates those, which are closer than a radius r. Finally, the result of distance computation is encoded in a d-dimensional binary vector: indexes with activated prototypes are marked with a '1' and the remaining with a '0'. The sparsity s of the encoded vector depends on the radius r. Alternatively, the sparsity can be kept constant with selective Kanerva coding (SKC) [22], where the $c = \lfloor s \cdot d \rfloor$ best matching prototypes are activated. This guarantees to preserve the sparsity independent of the query vector, but adds complexity due to the required sorting of d distances.

3.1.2 Randomized Activation Functions Encoding (RAFE): We propose a light-weight alternative to Kanerva coding, which relies only on one-dimensional distance computations. The key is that we compute a binary sub-vector for every feature value with a small SDM, which has d/n_f prototypes of dimension one. For every feature value $\mathbf{f}[k]$, we compute d/n_f binary values by activating the indexes, whose prototypes are within the radius r. Finally, all n_f sub-vectors are concatenated to form a d-dimensional binary vector (see Algorithm 1). This atomic vector (E) collectively represents the DVS events that are occurred during 0.05 s as shown in Fig. 1. The proposed encoding can be interpreted as a set of 0/1-valued activation functions with activation radius r centered around a random bias e_j; thus, we call it randomized activation functions encoding (RAFE). Thanks to the i.i.d. normal distribution of the bias e_j and the standardized features, the activation probability is normal too.

3.2 Combining Sparse Atomic Vectors to Build Prototypes

After encoding, multiple atomic vectors are combined to represent more complex structures by bundling all vectors belonging to the same ground truth velocity level. The bundling operation results in a *composite* vector that is similar to its constituent vectors and serves as a prototype to be stored in the AM. Separate prototypes are stored for x- and y-direction; in the following, we describe the methods without directional information for simplicity. The number of prototypes n_v depends on the velocities present in the training set. The ground truth velocity is quantized with a resolution of 0.002 m/s; the AM only stores prototypes for velocity levels encountered in the training set resulting in 19–266 number of prototypes. When using sparse binary vectors, the bundling operation simplifies to bitwise logical OR ($\vee$) [10, 16]. In this work, n_v velocity levels are represented by prototypes built as

$$P_i = \vee_{v_k = v_i} E_k \qquad i \in \{1, 2, ..., n_v\}, \tag{3}$$

Algorithm 1: RAFE

input : $\mathbf{f} \in \mathbb{R}^{n_f}$ - feature vector to be encoded
$e_1, e_2, ..., e_{d/n_f}$ $e_j \in \mathbb{R}$ - 1-d SDM memory
r - activation radius
output: $E \in \{0,1\}^d$ - encoded binary vector

for $\left(k = 1 : n_f\right)$ **do**
  Initialize $E_k = \mathbf{0}$
  for $\left(j = 1 : d/n_f\right)$ **do**
    $l \leftarrow \mathbf{f}[k] - e_j$
    if $-r \leq l \leq r$ **then**
      $E_k[j] \leftarrow 1$
    end
  end
  $E \leftarrow$ concatenate$(E_1, E_2, ..., E_{n_f})$
end

where v_k is the ground truth velocity of the encoded vector E_k, and v_i the velocity corresponding to prototype P_i.

During inference, a query vector Q is compared to every prototype according to the number of overlapping '1'; the velocity level with corresponding most similar prototype is the final estimated velocity. The overlap measure is a dot product in binary space, i.e., bitwise logical AND ($\wedge$) followed by pop-count:

$$\hat{v} = \underset{i \in \{1,2,\ldots,n_v\}}{\text{argmax}} |Q \wedge P_i|. \tag{4}$$

To obtain unbiased predictions, the density (i.e., the number of '1') of the prototypes in the AM should be equal. However, the density of a composite vector monotonically increases by bundling more atomic vectors. For instance, when a particular velocity label has too many examples, which obviously holds for the MVSEC dataset, a larger number of vectors are bundled which results in a higher density for the corresponding composite vector. Hence, this denser composite vector has a higher chance to be matched with any query. To cope with this problem, we apply context dependent thinning (CDT) [10] on composite vectors to preserve their sparsity. CDT consists of applying a random permutation (ρ) followed by a logical AND operation with the original vector. This core operation is repeated arbitrarily many times (here k times), applying different permutations. Finally, all intermediate results are ORed together:

$$Z = \vee_{i \leq k}(X \wedge \rho^i(X)) = X \wedge (\vee_{i \leq k} \rho^i(X)) \tag{5}$$

4 Online Model Update

Here, we propose a method for updating HD model during online operation to cope with the variability in the distribution of the DVS data due to various external conditions such as daylight, weather, or the environment. In most cases, these big variations are not represented by the training set, which degrades estimation performance in a dynamic environment. This demands to update the model at test-time based on predictions made by the model itself, also known as self-supervised learning.

Such online updates need to satisfy a few essential requirements to be effective and feasible for deployment on resource-limited devices. First, new observation should be added to the model only if the confidence of the prediction is sufficiently high. Therefore, an update is only done if the similarity to the best matching prototype exceeds a certain threshold. The threshold is determined with a cross-validated grid-search on the training set.

Second, the update of the model should not add significantly to the complexity nor to memory requirements, which depend on the representation of vectors. Here, sparse binary representations stand out allowing for computationally simple online updates. New observations (i.e., vectors) are directly added to the prototype with the OR operation. No additional intermediate composite vectors with integer counters are required for online updates; moreover, the operation is independent of the number of bundled vectors. CDT is applied if the density of the prototypes is getting too high, i.e., too many updates have been done.

In contrast, dense binary prototypes are built by element-wise majority vote, i.e., adding up all encoded vectors into a composite vector and binarize it based on a threshold, set to half of the number of added vectors. For online updates, the composite vector before binarization has to store all encoded vectors, requiring an additional memory that grows with the size of exemplar updates. A similar approach is made when representing prototypes with integer vectors, where the composite vector has to be rescaled by the number of accumulated vectors for simplified cosine similarity computation, requiring an additional integer counter per prototype.

Table 1: ARPE when directly encoding time-image, gradient-images, or global features to dense binary vectors. Prototypes are binarized (Dense Binary) or rescaled (Dense Integer) after training.

Features	Dense Binary	Dense Integer
Time-Image	0.1775	0.1651
Gradient-Images	0.1785	0.1771
6 Global Features	**0.1726**	**0.1632**

5 Experimental Results

This section assesses the proposed methods on the MVSEC dataset. We compare the average relative pose error (ARPE) defined as

$$\text{ARPE} = \frac{1}{n}\sum_{i=1}^{n} \arccos\left(\frac{< \hat{\mathbf{v}}_i, \mathbf{v}_i >}{||\hat{\mathbf{v}}_i||_2 \cdot ||\mathbf{v}_i||_2}\right), \tag{6}$$

where $\hat{\mathbf{v}}_i$ is the 2-D estimated velocity vector, $\mathbf{v}_i$ the ground truth velocity, and n the number of test samples in one sequence. The ARPE amounts to the average angular error between translational vectors while ignoring the scale [14]. For each of the five sequences in the dataset (i.e., recorder rides), a separate model is trained and tested, dividing the sequence into equally sized training (50%) and testing set (50%). Finally, the reported ARPE is the weighted average over all five sequences, taking the different lengths of the sequences into account.

5.1 Dense Binary Representations using Different Features

We first demonstrate the effectiveness of the six global features represented by dense binary vectors. The global features are compared to methods that omit the feature calculation, encoding either the time-image or gradient-images directly to binary vectors. Every pixel value of the time- or gradient-image is mapped to an atomic

Table 2: ARPE for different regression models on MVSEC dataset. SKC: selective Kanerva encoding; CDT: context dependent thinning; RAFE: randomized activation functions encoding; MLP: multi-layer perceptron.

Method	Dimension	Representation	Prototype	Similarity Measure	ARPE
Dense Binary [14]	$d = 8160$	Dense, Binary	Majority Gate	Hamming Distance	0.1726
Dense Integer	$d = 8160$	Dense, Integer	Addition	Cosine Similarity	0.1632
SKC [22]	$d = 8160$	Sparse, Binary	Logical OR	Overlap	0.1623
SKC+CDT	$d = 8160$	Sparse, Binary	Logocal OR + CDT	Overlap	0.1612
RAFE	$d = 8160$	Sparse, Binary	Logical OR	Overlap	0.1651
RAFE+CDT	$d = 8160$	Sparse, Binary	Logical OR + CDT	Overlap	**0.1593**
MLP	{6, 24, 80, 160, 130, 100, 75, 40, 12, 2}				**0.1564**
Linear Regression					0.1649
Regression degree 2					0.1615
Regression degree 3					0.1639

vector stored in a continuous item memory (CIM) [18], which is constructed such that pixel value k and $k + 1$ are similar (i.e., differ just by a small number of bits), and values 0 and $n - 1$ are orthogonal. Vectors are permuted according to the location in the image, and bundled together with all other encoded pixel-vectors. The permutation is done in x- and y-direction separately, whereas the permutation-map preserves similarity to neighboring locations, analogous to the CIM.

Table 1 compares the ARPE for different encoding methods with binary or integer prototypes. Encoding global features to dense binary vectors outperforms both time-image and gradient-images encoding. This supports the necessity of global feature extraction. Besides, the direct encoding of time-images or gradient-images requires a large amount of permutation and bundling operations in HD space, making these methods even more complex than first computing the features in low-dimensional space, and then encode them into a binary vector. Integer prototypes yield consistently lower ARPE than binary prototypes, which can be explained by their higher capacity. However, integer prototypes require larger memory footprint and cosine similarity calculation with integer arithmetic during inference, instead of Hamming distance computations with binary prototypes.

5.2 Encoding Methods using Global Features

Table 2 shows the ARPE for different HD regression models, compared with commonly used linear and polynomial regressions as well as a multi-layer perceptron (MLP), all operating on the six global features. The MLP has eight hidden layers (see Table 2) with ReLu activation, dropout, and batch normalization; it was highly optimized to have a small number of trainable parameters, while still being accurate.

Most sparse binary encoding schemes outperform the baseline dense encodings [14], with RAFE+CDT being the most accurate binary methods among the HD regressions; it has a 7.7% lower ARPE than dense binary encoding. RAFE+CDT is only outperformed by the deep MLP having full-precision, which has a marginal 1.8% lower ARPE. Classic polynomial regressions methods are less accurate than both MLP and RAFE+CDT, with the best polynomial regression of degree 2 being 3.3% and 1.4% worse than MLP and RAFE+CDT, respectively.

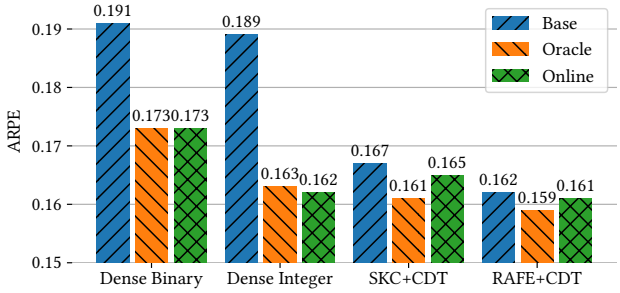

Figure 2: ARPE for online updates. Base model is trained on 25% of training data, the remaining 75% is used for updating model either based on ground truth labels (oracle) or on self generated labels (online).

5.3 Online updates

Fig. 2 shows the effect of online updates in both dense and sparse HD encodings. First, the model is trained on 25% of the training set, which is called the base model. The model is then updated on the remaining 75% on the training set either based on the ground truth label (oracle), or the label estimated by the model itself (online) given the confidence of the estimation is high enough. In all cases, the test set is not touched (i.e., no model updates) and stays the same as in previous experiments; therefore, the oracle results are the same as reported in Table 2. As shown, in all encodings, the ARPE drops when doing updates, and sparse encodings consistently achieve lower ARPE than the dense encodings. The self-supervised online updates with RAFE+CDT achieves lowest ARPE (0.161), almost the same as its oracle (0.159). This means when RAFE+CDT uses only 25% of training data followed by online updates, it will result in similar ARPE compared to the model trained with 100% of data. Further, it allows the model update without any additional memory during inference.

5.4 Embedded Accelerator and Comparisons

Finally, the inference of the most successful encodings are deployed on GAP8 [5], a RSIC-V based ultra-low power commercial SoC containing an 8-core computational cluster. The dimension of all binary encodings is reduced to d=2880 since it yielded negligible

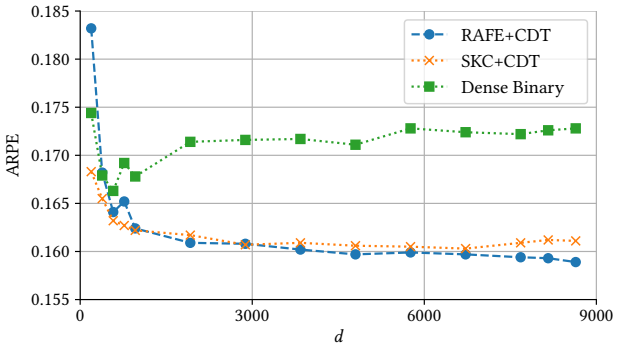

Figure 3: ARPE vs. dimension d of the binary vectors.

Table 3: Comparing methods using pre-computed global features on GAP8 accelerator with d=2880 during inference.

Method	ARPE	T./Inf.	Power	En./Inf.
Dense Binary[14]	0.1716	**0.28 ms**	18.5 mW	5.2 μJ
SKC+CDT	0.1607	17.98 ms	9.5 mW	170.8 μJ
RAFE+CDT	0.1608	**0.28 ms**	17.9 mW	**5.0 μJ**
MLP	**0.1575**	1.75 ms	28.1 mW	49.2 μJ

increase in ARPE (see Fig. 3), in the dense binary case it even improved due to quantization effects with thermometer mapping. The MLP is mapped with the open-source toolkit FANN-on-MCU [23], that uses fixed-point arithmetic that also negligibly increases ARPE to 0.1575. Table 3 compares the inference time, power consumption, and energy per inference for encoding and classification of the six global features. These measurements are done at 1.2 V and 100 MHz, the frequency at which the board has demonstrated to be most efficient. All methods are computed in 32-bit fixed-point arithmetic and parallelized on the 8-core cluster using OpenMP directives, where a maximum parallel speedup of 6.9× was achieved for RAFE+CDT. Data encoding is executed in parallel with static scheduling and using 32-dimensional batches to avoid data dependencies and memory conflicts. Outer loops are further unrolled to permit hard-coding parameters and save unnecessary computation. The same approach is taken in the inference process with parallel comparison between the query vector and the SDM prototypes. Thanks to the small memory requirements of 2880-dimensional vectors, all data is stored on the fast cluster memory that allows over 2× speedup compared to the larger on-chip memory. RAFE+CDT outperforms all other encodings with the lowest latency of 0.28 ms and energy of 5.0 μJ per inference, being 9.84× more energy-efficient and 6.25× faster than MLP at the cost of 2.1% ARPE increase. The significant improvement of RAFE+CDT compared to SKC+CDT comes from the distance computation (1-D distance instead of 6-D ℓ_2-norm) and the simplified activation function (feed-forward activation instead of sorting operation). Due to similar operations in encoding and classification, dense binary performs on par with RAFE+CDT, however, at 6.7% higher ARPE.

6 Conclusion

This work demonstrates the use of sparse binary representations in regression tasks for event-based DVS. The novel randomized activation functions encoding is among the most accurate regression methods, and allows for accurate online model update closely matching oracle, while staying fully in binary space without any additional memory. In an 8-core accelerator, it achieves 9.84× higher energy efficiency than a MLP at comparable ARPE (0.1608 vs. 0.1575).

Acknowledgments

This project was supported in part by ETH Research Grant 09 18-2, and by EU's H2020 under grant no. 780215.

References

[1] A. Amir, et al. 2017. A Low Power, Fully Event-Based Gesture Recognition System. In *2017 IEEE CVPR*. IEEE, 7388–7397.
[2] C. Brandli, et al. 2014. A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor. *IEEE Journal of Solid-State Circuits* 49, 10 (2014), 2333–2341.
[3] Nicola Cottini, et al. 2012. A 33μW 42 GOPS/W 64x64 Pixel Vision Sensor with Dynamic Background Subtraction for Scene Interpretation. In *2012 ACM/IEEE ISLPED*. ACM Press, New York, New York, USA, 315–320.
[4] C. Eliasmith. 2013. *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford Series on Cognitive Models and Architectures.
[5] E. Flamand, et al. 2018. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In *2018 IEEE 29th ASAP*. 1–4.
[6] G. Gallego, et al. 2018. A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation. *2018 IEEE CVPR* (2018), 3867–3876.
[7] P. Kanerva. 1988. *Sparse distributed memory*. MIT Press, Cambridge, Massachusetts.
[8] P. Kanerva. 2009. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation* 1, 2 (2009), 139–159.
[9] Geethan Karunaratne, et al. 2020. In-memory hyperdimensional computing. *Nature Electronics* (01 Jun 2020).
[10] D. Kleyko, et al. 2018. Classification and Recall With Binary Hyperdimensional Computing: Tradeoffs in Choice of Density and Mapping Characteristics. *IEEE Transactions on Neural Networks and Learning Systems* 29, 12 (2018), 5880–5898.
[11] Ana I Maqueda, et al. 2018. Event-based vision meets deep learning on steering prediction for self-driving cars. In *2018 IEEE CVPR*. IEEE, 5419–5427.
[12] M. Mendes, et al. 2008. Robot navigation using a sparse distributed memory. In *2008 IEEE ICRA*. IEEE, 53–58.
[13] Anton Mitrokhin, et al. 2018. Event-Based Moving Object Detection and Tracking. In *2018 IEEE/RSJ IROS*. IEEE, 1–9.
[14] A. Mitrokhin, et al. 2019. Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception. *Science Robotics* 4, 30 (5 2019), eaaw6736.
[15] P. Neubert, et al. 2019. An Introduction to Hyperdimensional Computing for Robotics. *KI - Künstliche Intelligenz* (2019), 319—-330.
[16] D. A. Rachkovskij. 2001. Representation and processing of structures with binary sparse distributed codes. *IEEE Transactions on Knowledge and Data Engineering* 13, 2 (2001), 261–276.
[17] A. Rahimi, et al. 2017. High-Dimensional Computing as a Nanoscalable Paradigm. *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, 9 (2017), 2508–2521.
[18] A. Rahimi, et al. 2019. Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals. *Proc. IEEE* 107, 1 (2019), 123–143.
[19] A. Rahimi, et al. 2016. A Robust and Energy-Efficient Classifier Using Brain-Inspired Hyperdimensional Computing. In *2016 ACM/IEEE ISLPED*. ACM Press, New York, New York, USA, 64–69.
[20] A. Renner, et al. 2019. Event-Based Attention and Tracking on Neuromorphic Hardware. In *IEEE CVPR Workshops*. IEEE.
[21] B. Son, et al. 2017. A 640×480 dynamic vision sensor with a 9µm pixel and 300Meps address-event representation. In *2017 IEEE ISSCC*. IEEE, 66–67.
[22] J. B. Travnik et al. 2017. Representing high-dimensional data to intelligent prostheses and other wearable assistive robots: A first comparison of tile coding and selective Kanerva coding. In *2017 IEEE ICORR*. IEEE, 1443–1450.
[23] Xiaying Wang, et al. 2020. FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things. *IEEE Internet of Things Journal* 7, 5 (2020), 4403–4417.
[24] A. Z. Zhu, et al. 2018. The Multivehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception. *IEEE Robotics and Automation Letters* 3, 3 (2018), 2032–2039.