

Hyperseed: Unsupervised Learning with Vector Symbolic Architectures

Evgeny Osipov, Sachin Kahawala, Dilantha Haputhanthri, Thimal Kempitiya, Daswin De Silva, Damminda Alahakoon, Denis Kleyko

Abstract—Motivated by recent innovations in biologically-inspired neuromorphic hardware, this article presents a novel unsupervised machine learning algorithm named Hyperseed that draws on the principles of Vector Symbolic Architectures (VSA) for fast learning of a topology preserving feature map of unlabelled data. It relies on two major operations of VSA, binding and bundling. The algorithmic part of Hyperseed is expressed within Fourier Holographic Reduced Representations model, which is specifically suited for implementation on spiking neuromorphic hardware. The two primary contributions of the Hyperseed algorithm are, few-shot learning and a learning rule based on single vector operation. These properties are empirically evaluated on synthetic datasets as well as on illustrative benchmark use-cases, IRIS classification, and a language identification task using n -gram statistics. The results of these experiments confirm the capabilities of Hyperseed and its applications in neuromorphic hardware.

Index Terms—self-organizing maps, vector symbolic architectures, hyperseed, neuromorphic hardware

I. INTRODUCTION

Vector Symbolic Architectures (VSA) are increasingly leveraged and adapted in machine learning and robotics algorithms and applications [1]–[6]. In classification tasks, the use of VSA leads to order of magnitude increase in energy efficiency of computations on the one hand and natively enables one-shot and multi-task learning on the other [7]–[11]. It is prospected that VSA will play a key role in the development of novel neuromorphic computer architectures [12] as an algorithmic abstraction [13], [14]. The main contribution of this article is a novel algorithm for unsupervised learning

Manuscript received on October 19, 2021; revised on June 30, 2022; accepted on September 18, 2022. This work was supported in part by the Intel Neuromorphic Research Community Grant to Luleå University of Technology, the Swedish Foundation for international Cooperation in Research and Higher Education (STINT) under Mobility Grant for Internationalisation MG2020-8842, and the Russian Science Foundation during the period of 2020-2021 under grant 20-71-10116. SK and DH are recipients of the Centre for Data Analytics and Cognition (CDAC) Ph.D. research scholarships. DK has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 839179.

E. Osipov is with the Department of Computer Science, Electrical and Space Engineering at Luleå University of Technology, 97187 Luleå, Sweden. E-mail: Evgeny.Osipov@ltu.se

S. Kahawala, D. Haputhanthri, T. Kempitiya, D. De Silva and D. Alahakoon are with the Centre for Data Analytics and Cognition (CDAC) at La Trobe University, Melbourne, Australia. E-mail: {S.Kahawala, D.Haputhanthri, T.Kempitiya, D.DeSilva, D.Alahakoon}@latrobe.edu.au

D. Kleyko is with the Redwood Center for Theoretical Neuroscience at the University of California, Berkeley, CA 94720, USA and also with the Intelligent Systems Lab at Research Institutes of Sweden, 16440 Kista, Sweden. E-mail: denis.kleyko@ri.se

called Hyperseed, which relies on the mathematical properties arising from random high-dimensional representation spaces described through the phenomenon of the concentration of measure [15] and of the main VSA operations of binding and superposition [16]. The method’s name suggests that data samples are encoded as high-dimensional vectors (also called *hypervectors*, HVs or “seeds”), which are then mapped (a.k.a. “sowed”) onto a specially prepared topologically arranged set of hypervectors for revealing the internal cluster structure in the unlabeled data.

The Hyperseed algorithm bears conceptual similarities to Kohonen’s Self-Organizing Maps (SOM) algorithm [17], [18], therefore, selected SOM terminology is adopted for the description of our approach. However, it is designed using cardinality different computing principles to the SOM algorithm. Hyperseed implements the entire learning pipeline in terms of VSA operations. To the best of our knowledge, this has not been attempted prior and is reported for the first time.

The Hyperseed algorithm is presented using Frequency Holographic Reduced Representations (FHRR) [19] model of VSAs and the concept of Fractional Power Encoding (FPE) [19]–[23]. The usage of the FHRR model makes the proposed solution specifically fit for implementation on spiking neural network architectures including Intel’s Loihi [12].

To this end, we introduce the Hyperseed algorithm and demonstrate its performance on three illustrative non-linear classification problems of varying complexity: Synthetic datasets from the Fundamental Clustering Problems Suite, Iris classification, and language identification using n -gram statistics. Across all experiments, Hyperseed convincingly demonstrates its key novelties of learning from a few input vectors and single vector operation learning rule, both of which contribute towards reduced time and computation complexity.

The article is structured as follows. Section II describes related work to Hyperseed operations. The VSA methods leveraged in Hyperseed are presented in Section III. Section IV presents the main contribution – the method for unsupervised learning – Hyperseed. Section V reports the results of the experimental performance evaluation. Section VI discusses the suitability of Hyperseed for realization on neuromorphic hardware. The conclusions follow in Section VII.

II. RELATED WORK

VSA [19], [24]–[26] is a computing framework providing methods of representing and manipulating concepts and their meanings in a high-dimensional space. VSA finds its

applications in, for example, cognitive architectures [27]–[29], natural language processing [30]–[34], communications [35]–[37], biomedical signal processing [1], [38], approximation of conventional data structures [39], [40], and for classification tasks such as gesture recognition [1], [41], cybersecurity threat detection [42], [43], physical activity recognition [44], character recognition [45], [46], speaker identification [47], fault isolation and diagnostics [48]–[50]. Examples of efforts on using VSA for other than classification learning tasks are using data HVs for clustering [51]–[53], semi-supervised learning [54], collaborative privacy-preserving learning [55], [56], multi-task learning [7], [8], distributed learning [57], [58]. A comprehensive two-part survey of VSA is available in [59], [60].

Hypervectors of high (but fixed) dimensionality (denoted as d) are the basis for representing information in VSA. The information is distributed across hypervector positions, therefore, hypervectors use distributed representations [61]. There are different VSA models that all offer the same operation primitives but differ slightly in terms of the implementation of these primitives. For example, there are VSA models that compute with binary, bipolar [62], [63], continuous real, and continuous complex vectors [19]. Thus, the VSA concept has the flexibility to connect to a multitude of different hardware types, such as binary-valued VSAs for analog in-memory computing architectures [9] or complex-valued VSAs for spiking neuron architectures [64]–[66].

The relevant sub-domain of related work to the proposed Hyperseed algorithm is the application of VSA for solving machine learning tasks. In this context, VSA have been used for: 1) Representing input data and interfacing such representations with conventional machine learning algorithms and 2) Implementing the functionality of neural networks with VSA operations.

The most illustrative use cases for encoding of input data into hypervectors and interfacing conventional machine learning algorithms are [52], [67]–[75]. For example, works [69], [71] proposed encoding n -gram statistics into hypervectors and subsequently solving typical natural language processing tasks with either supervised or unsupervised learning using standard artificial neural network architectures. The main distinctive property of VSA represented data is the substantial reduction of the memory footprint and the reduced learning time. In [52], hypervectors were used to encode sequences of variable lengths in the context of unsupervised learning of traffic patterns in intelligent transportation system application. In the context of visual navigation, hypervectors were used as input to Simultaneous Localization and Mapping (SLAM) algorithms [3] as well as for ego-motion estimation [76]–[78].

A great potential of VSAs was demonstrated when used for the implementation of the entire functionality of some classical neural network architectures. In [5], [79], [80] the functionality of an entire class of randomly connected neural networks (random vector functional link networks [81] and echo state networks [82]) was implemented purely in terms of VSA operations. It was demonstrated that implementing the algorithm functionality with bipolar VSAs allows reducing energy consumption on the specialized digital hardware by the order

of magnitude, while substantially decreasing the operation times. Moreover, further flexibility can be achieved [50], [83] when considering the ways of generating random connections used in the networks.

The main contribution of this article in the context of VSA is a novel approach to learning since the dominating learning approach in the area is on creating a single hypervector for a specific class. This is achieved through encoding input data and then forming associative memory storing the prototypical representations for individual classes. Our approach to learning is radically different – it utilizes the similarity preservation property of the binding operation in combination with the FPE encoding method [19]. FPE was recently used to simulate and predict dynamical systems [84], perform integer factorization [85], and represent order in time series [86]. The associative memory in the proposed approach is created once during the initialization phase and remains fixed during the life-time of the system. The update requires a single vector operation. To the best of our knowledge, this is the first research article to present the usage of VSA in unsupervised learning tasks.

III. METHOD: HOLOGRAPHIC REDUCED REPRESENTATIONS (HRR) MODEL

The Hyperseed algorithm is designed using the Fourier Holographic Reduced Representations (FHRR) model [19]. FHRR facilitates the mathematical treatment of Hyperseed operations. The potential argument that complex numbers used in FHRR add to the memory requirements of Hyperseed is intuitively true in the case of CPU realization. However, in Section VI, we rationalise this is not an issue for the neuromorphic hardware. Also, due to the equivalence of FHRR and HRR models, the operations of Hyperseed can be implemented with hypervectors from \mathbb{R}^d . In fact, when evaluating the performance of the bottleneck functionality of Hyperseed on Intel’s Loihi, we use HRR model¹. The atomic FHRR hypervectors are randomly sampled from \mathbb{C}^d . Dimensionality d is a hyperparameter of Hyperseed. In high-dimensional random spaces, all random hypervectors are dissimilar to each other (quasi-orthogonal) with an extremely high probability. VSA defines operations and a similarity measure on hypervectors. In this article, we use the cosine similarity of real parts of hypervectors for characterizing the similarity. The three primary operations for computing with hypervectors are superposition, binding, and permutation.

A. Binding operation

The binding operation is used to bind two hypervectors together. The result of the binding is another hypervector. For example, for two hypervectors \mathbf{v}_1 and \mathbf{v}_2 the result of binding of their hypervectors (denoted as \mathbf{b}) is calculated as follows:

$$\mathbf{b} = \mathbf{v}_1 \circ \mathbf{v}_2, \quad (1)$$

where the notation \circ is used to denote the binding operation. In HRR, the binding operation is implemented as circular

¹The supplementary code base also contains the HRR implementation of the algorithm.

convolution of \mathbf{v}_1 and \mathbf{v}_2 , which can be implemented as the component-wise multiplication in the Fourier domain. This observation inspired FHRR where the representations are already in the Fourier domain in a form of phasors so that the component-wise multiplication, which is equivalent to the addition of phase angles modulo 2π , plays the role of the binding operation. Binding is, essentially, a randomizing operation that moves hypervectors to another (random) part of the high-dimensional space. The role played by the binding operation depends on the algorithmic context. In data structures with roll-filler pairs, the binding operation corresponds to the assignment of a value (filler) to a variable (role). There are two important properties of the binding operation. First, the resultant hypervector \mathbf{b} is dissimilar to the hypervectors being bound, i.e., the similarity between \mathbf{b} and \mathbf{v}_1 or \mathbf{v}_2 is approximately 0.

Second, the binding operation preserves similarity. That is the distribution of the similarity measure between hypervectors from some set \mathcal{S} is preserved after binding of all hypervectors in \mathcal{S} with the same random hypervector \mathbf{v} .

The binding operation is reversible. The unbinding, denoted as \oslash , is implemented by the circular correlation in HRR. In the case of FHRR this is equivalent to component-wise multiplication with the complex conjugate. Being the inverse of the binding operation, the unbinding obviously has the same similarity preservation property when performed on all hypervectors in \mathcal{S} with the same hypervector \mathbf{v} :

$$\mathbf{v}_2 \oslash \mathbf{b} = \mathbf{v}_1. \quad (2)$$

The interpretation of the unbinding operation is a retrieval of a value from the hypervector encoding the assignment. When unbinding is performed from the superposition of bindings (see Section III-C), the retrieved hypervector contains noise. In VSA, the noisy vector can be cleaned-up by performing a search for the closest atomic hypervector stored in an associative memory.

B. Permutation operation

The permutation (rotation) operation $\mathbf{b} = \rho(\mathbf{v})$ is a unitary operation that is commonly used to represent an order of the symbol in a sequence. As with the binding operation, the resultant hypervector \mathbf{b} is dissimilar to \mathbf{v} . In this article, this operation is used for encoding a certain type of input data as further described in Section V.

C. Superposition operation

Superposition is denoted with $+$ and implemented via component-wise addition. The superposition operation combines several hypervectors into a single hypervector. For example, for hypervectors \mathbf{v}_1 and \mathbf{v}_2 the result of superposition (denoted as \mathbf{a}) is simply:

$$\mathbf{a} = \mathbf{v}_1 + \mathbf{v}_2. \quad (3)$$

In contrast to the binding operation, the resultant hypervector \mathbf{a} is similar to all superimposed hypervectors, i.e., the cosine similarity between \mathbf{b} and \mathbf{v}_1 or \mathbf{v}_2 is larger than 0. If several

copies of any hypervector are included (e.g., $\mathbf{a} = 3\mathbf{v}_1 + \mathbf{v}_2$), the resultant hypervector is more similar to the dominating hypervector than to other components.

If superposition is applied to several bindings it is possible to unbind any hypervector from any binding. In this case, the result of the unbinding operation is a noisy version of the second operand of the particular binding. For example, if $\mathbf{a} = \mathbf{v}_1 \circ \mathbf{v}_2 + \mathbf{u}_1 \circ \mathbf{u}_2$, then $\mathbf{u}_2 \oslash \mathbf{a} = \mathbf{u}_1 + \text{noise} = \mathbf{u}_1^*$. Given that noiseless atomic hypervectors ($\mathbf{v}_1, \mathbf{v}_2, \mathbf{u}_1, \mathbf{u}_2$) are kept in the associative memory and so vector \mathbf{u}_1^* is expected to have the highest similarity to \mathbf{u}_1 . The same property holds for the binding of any atomic hypervector with the superposition of unbindings (which we use below in the description of our approach). That is if $\mathbf{a} = \mathbf{v}_1 \oslash \mathbf{v}_2 + \mathbf{u}_1 \oslash \mathbf{u}_2$, then $\mathbf{u}_2 \circ \mathbf{a} = \mathbf{u}_1 + \text{noise} = \mathbf{u}_1^*$.

IV. HYPERSEED: UNSUPERVISED LEARNING WITH VECTOR SYMBOLIC ARCHITECTURES

This section presents the main contribution of this article – the method for unsupervised learning – Hyperseed. Denote the set of FHRR-represented input data as $\mathcal{D} \in \mathbb{C}^d$. Data hypervectors for training are generated during the encoding phase (see Section V-A for the details of the encoding). They are kept in a working memory. The input query for testing is also a data hypervector obtained using the same encoding procedure as for the training data.

Denote as $\mathcal{P} \in \mathbb{C}^d$ the vector space with known similarity properties. Set \mathcal{P} is created by encoding points (also referred to as nodes further in the text) of a 2D grid using FPE method [19]–[23]. The cardinality of HD-map $|\mathcal{P}| = n \times m$, where n and m are the sizes of the grid along the vertical and the horizontal axes, respectively. For the sake of brevity, further in the text we will refer to set \mathcal{P} as HD-map. HD-map is computed, as described below, once and is stored in the associative memory. This memory is fixed throughout the life-time of the system.

The Hyperseed algorithm relies on the similarity preservation property of the (un)binding operation. The goal with Hyperseed is to translate the original data hypervectors \mathcal{D} (with unknown) internal similarity layout to HD-map \mathcal{P} by unbinding all of its members from hypervector \mathbf{s} , i.e.:

$$\mathcal{D} \oslash \mathbf{s} \Rightarrow \mathcal{P}. \quad (4)$$

Hypervector \mathbf{s} is obtained as the result of applying an unsupervised learning rule. In essence, during the learning, some selected hypervectors from \mathcal{D} will be bound to selected vectors from \mathcal{P} as described further².

A. Initialization Phase: Generation of HD-map \mathcal{P} and hypervector \mathbf{s}

The hypervectors, which are the members of \mathcal{P} , are generated such that the similarity between them relates to topological proximity of grid nodes. Note, however, that the reference to the topological arrangement of \mathcal{P} is virtual in a sense that

²Due to an analogy of “seeding” data hypervectors onto HD-map, hypervector \mathbf{s} is referred as seed vector throughout the article.

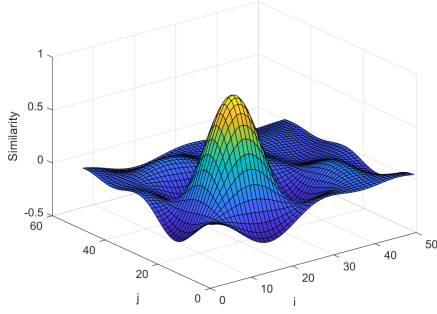


Fig. 1: Similarity distribution on an HD-map. The target node is (15, 15), the size of the grid is 50×50 , bandwidth $\epsilon = 0.05$.

in the associative memory, in which hypervectors of \mathcal{P} are stored does not have any structure. Topology information is kept on a side to be used for visualization purposes only.

The generation of HD-map starts with two randomly generated unit hypervectors $\mathbf{x}_0, \mathbf{y}_0 \in \mathbb{C}^d$ as $\mathbf{x}_0 \sim e^{j \cdot 2\pi \cdot U(0,1)}$ and $\mathbf{y}_0 \sim e^{j \cdot 2\pi \cdot U(0,1)}$. Let us denote the bandwidth parameter regulating the similarity between the adjacent coordinates on the grid by ϵ . The i -th x and y coordinates of the grid will be created using the FPE method as:

$$\mathbf{x}_i = \mathbf{x}_0^{\epsilon \cdot i}, \mathbf{y}_i = \mathbf{y}_0^{\epsilon \cdot i}. \quad (5)$$

The hypervector $\mathbf{p}_{(i,j)}$ representing a node with coordinates (i, j) on the grid is computed as $\mathbf{p}_{(i,j)} = \mathbf{x}_i \circ \mathbf{y}_j$. Fig. 1 illustrates the landscape of similarity between all hypervectors of HD-map stored in the associative memory and one selected hypervector of the same HD-map encoding coordinates (15,15) on 50×50 2D grid as a function of coordinates i and j .

Hypervector \mathbf{s} is also initialized randomly: $\mathbf{s} \sim e^{j \cdot 2\pi \cdot U(0,1)}$. It is updated over several iterations during the learning phase as described below. The number of update iterations is a hyperparameter of Hyperseed.

B. Search procedure in Hyperseed: Finding Best Matching Vector on HD-map

In the Hyperseed algorithm, HD-map \mathcal{P} acts as the auto-associative memory [87]–[89]. That is the only operation performed on HD-map is the search for the Best Matching Vector (BMV) given some input hypervector. The BMV is found by computing the cosine similarity between the input hypervector and all hypervectors in \mathcal{P} . The output of this procedure is a valid hypervector in \mathcal{P} with the highest similarity to the input hypervector.

The mapping ($\mathbf{d}_i \rightarrow \mathbf{p}_i$) of data hypervectors in \mathcal{D} to hypervectors of HD-map \mathcal{P} is done by unbinding \mathbf{d}_i from the trained hypervector \mathbf{s} :

$$\mathbf{p}_i^* = \mathbf{d}_i \oslash \mathbf{s}. \quad (6)$$

In (6), \mathbf{p}_i^* is a noisy version of a hypervector in \mathcal{P} .

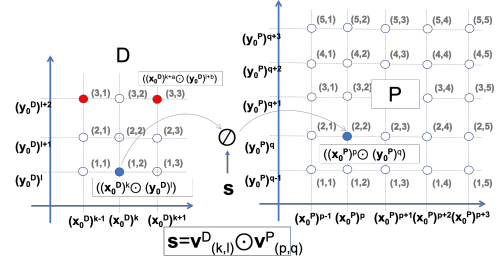


Fig. 2: Transformation of space \mathcal{D} into \mathcal{P} with the unbinding operation.

C. Update phase: Unsupervised learning of hypervector \mathbf{s}

The goal with the update procedure on each iteration is to map input hypervector \mathbf{d}_i as near as possible to some target hypervector in \mathcal{P} with respect to the cosine similarity.

Therefore, a single learning iteration consists of three steps:

- 1) Choose a target hypervector $\mathbf{p}_{\text{target}}$ (see the next subsection);
- 2) Compute a hypervector for the perfect mapping $\mathbf{d}_i \rightarrow \mathbf{p}_{\text{target}}$ by binding of \mathbf{d}_i with $\mathbf{p}_{\text{target}}$.
- 3) Update hypervector \mathbf{s} by adding this perfect mapping hypervector to hypervector \mathbf{s} :

$$\mathbf{s} = \mathbf{s} + \mathbf{d}_i \circ \mathbf{p}_{\text{target}}. \quad (7)$$

Note that after the update, \mathbf{s} is not a phasor vector anymore so it might be renormalized if necessary. Thus, by the end of the learning phase, hypervector \mathbf{s} is the superposition of bindings $\mathbf{d}_i \circ \mathbf{p}_j$. As such, the result of unbinding of hypervectors similar to \mathbf{d}_i with hypervector \mathbf{s} (6) will resemble hypervectors in \mathcal{P} (i.e., the hypervectors used in the update phase).

D. Weakest match search (WMS) phase: Finding a data hypervector for the update in a single iteration

To find a data hypervector for the update of hypervector \mathbf{s} (7), Hyperseed uses a heuristic based on the farthest-first traversal rule (FFTR). This principle is widely used for defining heuristics in many important computing applications ranging from approximation of Traveling Salesman Problem [90] to k -center clustering [91] and fast similarity search [92]. FFTR has been also used as a weight update rule in SOMs [93], which resulted in better representation of outliers as well as lower topographic and quantization errors. In FFTR, the first point is selected arbitrarily and each successive point is as far as possible from the set of previously selected points. In the case of Hyperseed, FFTR is straightforwardly implemented by checking the cosine similarity between the noisy vector \mathbf{p}^* (6) with all noiseless hypervectors of HD-map \mathcal{P} . In order to demonstrate this, we shall consider the properties of the transformation from \mathcal{D} to \mathcal{P} with the unbinding operation (4).

Consider an example of transforming vector space \mathcal{D} of FPE-encoded two dimensional grid structure to HD-map \mathcal{P} . Graphically, the scenario is illustrated in Fig. 2. The data is transformed via FPE as described in Section IV-A with initial random bases \mathbf{x}_0^D and \mathbf{y}_0^D . Importantly, while the

transformed hypervectors in \mathcal{D} are available for observation, the base hypervectors, which were used in the transformation are unknown. The bandwidth $\epsilon_{\mathcal{D}}$ used during the encoding is also unknown.

Vector space \mathcal{P} is represented via FPE, now with initial random bases $\mathbf{x}_0^{\mathcal{P}}$ and $\mathbf{y}_0^{\mathcal{P}}$. Importantly, the base hypervectors of \mathcal{P} as well as FPE bandwidth $\epsilon_{\mathcal{P}}$ are known. For brevity of calculations, it is assumed the FPE bandwidth of vector space \mathcal{D} is the same as that of vector space \mathcal{P} , that is $\epsilon_{\mathcal{P}} = \epsilon_{\mathcal{D}} = \epsilon$.

We now pick an arbitrary vector from \mathcal{D} encoding a pair of values (k, l) and bind it to a hypervector from \mathcal{P} encoding some predefined pair of values (p, q) :³

$$\begin{aligned} \mathbf{s} &= e^{j \cdot 2\pi\epsilon_{\mathcal{P}} \cdot \mathbf{x}_0^{\mathcal{P}} \cdot p + j \cdot 2\pi\epsilon_{\mathcal{P}} \cdot \mathbf{y}_0^{\mathcal{P}} \cdot q + j \cdot 2\pi\epsilon_{\mathcal{D}} \cdot \mathbf{x}_0^{\mathcal{D}} \cdot k + j \cdot 2\pi\epsilon_{\mathcal{D}} \cdot \mathbf{y}_0^{\mathcal{D}} \cdot l} \\ &= e^{j \cdot 2\pi\epsilon_{\mathcal{P}} \cdot (\mathbf{x}_0^{\mathcal{P}} \cdot p + \mathbf{y}_0^{\mathcal{P}} \cdot q + \frac{\epsilon_{\mathcal{D}}}{\epsilon_{\mathcal{P}}} \mathbf{x}_0^{\mathcal{D}} \cdot k + \frac{\epsilon_{\mathcal{D}}}{\epsilon_{\mathcal{P}}} \mathbf{y}_0^{\mathcal{D}} \cdot l)}. \end{aligned} \quad (8)$$

Obviously, as the result of unbinding of the hypervector representing the coordinate (k, l) from \mathcal{D} from hypervector \mathbf{s} , it will be translated to the hypervector for (p, q) from \mathcal{P} as illustrated in Fig. 2.

Let us now unbind a hypervector from \mathcal{D} encoding a point on a certain offset (a, b) from (k, l) , that is hypervector: $e^{j \cdot 2\pi\epsilon_{\mathcal{D}} \cdot (\mathbf{x}_0^{\mathcal{D}} \cdot (k+a) + \mathbf{y}_0^{\mathcal{D}} \cdot (l+b))}$, which is the farthest from point (k, l) in this scenario. Recall that unbinding in FHRR is implemented as a component-wise multiplication with the complex conjugate:

$$\begin{aligned} \mathbf{v}^* &= e^{j \cdot 2\pi\epsilon_{\mathcal{P}} \cdot (\mathbf{x}_0^{\mathcal{P}} \cdot p + \mathbf{y}_0^{\mathcal{P}} \cdot q + \frac{\epsilon_{\mathcal{D}}}{\epsilon_{\mathcal{P}}} \mathbf{x}_0^{\mathcal{D}} \cdot k + \frac{\epsilon_{\mathcal{D}}}{\epsilon_{\mathcal{P}}} \mathbf{y}_0^{\mathcal{D}} \cdot l)} \\ &\quad \cdot e^{j \cdot 2\pi\epsilon_{\mathcal{D}} \cdot (-\mathbf{x}_0^{\mathcal{D}} \cdot (k+a) - \mathbf{y}_0^{\mathcal{D}} \cdot (l+b))} \\ &= e^{j \cdot 2\pi\epsilon_{\mathcal{P}} \cdot (\mathbf{x}_0^{\mathcal{P}} \cdot (p + \alpha_1 \cdot a) + \mathbf{y}_0^{\mathcal{P}} \cdot (q + \alpha_2 \cdot b))}. \end{aligned} \quad (9)$$

In 9, $\alpha_1 = -\frac{\epsilon_{\mathcal{D}} \cdot \mathbf{x}_0^{\mathcal{D}}}{\epsilon_{\mathcal{P}} \cdot \mathbf{x}_0^{\mathcal{P}}}$ and $\alpha_2 = -\frac{\epsilon_{\mathcal{D}} \cdot \mathbf{y}_0^{\mathcal{D}}}{\epsilon_{\mathcal{P}} \cdot \mathbf{y}_0^{\mathcal{P}}}$ are coefficients introduced in order to align the result of unbinding with HD-map \mathcal{P} for ease of the interpretation.

In α_1 and α_2 , the parameter of interest is $\epsilon_{\mathcal{P}}$, which is the bandwidth of HD-map \mathcal{P} and is the hyperparameter of the algorithm. Consider the case where $\epsilon_{\mathcal{P}} > \epsilon_{\mathcal{D}}$. This means that the fidelity of the similarity between hypervectors in HD-map \mathcal{P} is much coarser compared to the fidelity of the interhypervector similarity in the original vector space \mathcal{D} . In this case, all hypervectors from \mathcal{D} after unbinding will be similar to the hypervector in \mathcal{P} , which was chosen for the update of seed hypervector \mathbf{s} (e.g., $\mathbf{v}_{(p,q)}^{\mathcal{P}}$ in Fig. 2). Essentially, we will observe an effect of collapsing of all hypervectors in space \mathcal{D} onto a single hypervector from space \mathcal{P} . This is demonstrated by a simulation in which hypervector \mathbf{s} was created as $\mathbf{s} = \mathbf{v}_{(1,2)}^{\mathcal{D}} \circ \mathbf{v}_{(2,2)}^{\mathcal{P}}$. The size of HD-map \mathcal{P} is 5×5 . Two simulations were performed: 1.) With the FPE bandwidth for encoding input data being equal 0.2, while the FPE bandwidth of HD-map was set to 0.03; and 2.) With the FPE bandwidth for encoding input data being equal 0.2, while the FPE bandwidth of HD-map was set to 0.8. Fig. 3a and 3b show the distribution of cosine similarities to all hypervectors of HD-map for every data hypervector after unbinding with hypervector \mathbf{s} (6) for the first and second simulation, respectively.

³Strictly speaking, equations below should include modulo 2π operations but they are omitted for the sake of readability.

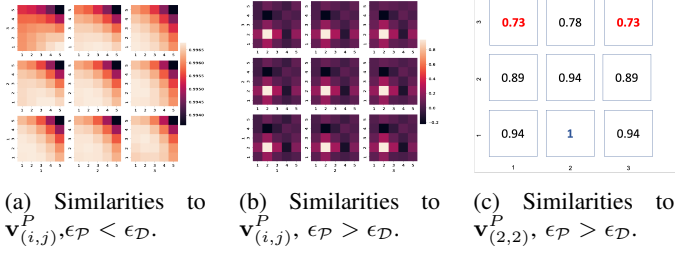


Fig. 3: Distribution of cosine similarities to hypervectors in \mathcal{P} for every data hypervector after unbinding with hypervector \mathbf{s} .

Fig. 3b demonstrates the effect of collapsing of all points in \mathcal{P} onto hypervector $\mathbf{v}_{(2,2)}^{\mathcal{P}}$. Fig. 3c shows cosine similarities for every data hypervector after unbinding with hypervector \mathbf{s} to the BMV in the second simulation (i.e., $\mathbf{v}_{(2,2)}^{\mathcal{D}}$). Observe that the lowest similarities are for hypervectors $\mathbf{v}_{(1,3)}^{\mathcal{D}}$ and $\mathbf{v}_{(3,3)}^{\mathcal{D}}$, which are the farthest away from the hypervector $\mathbf{v}_{(1,2)}^{\mathcal{D}}$ used to compute hypervector \mathbf{s} . Therefore, following the FFTR heuristic, one of these hypervectors should be used in (7) to update hypervector \mathbf{s} , thus, creating a new point of attraction in \mathcal{P} . To summarize, the WMS procedure is as follows:

- 1) Initialize the lowest similarity variable $D_{min} = 1$;
- 2) For each hypervector in \mathcal{D} compute p^* (6) and search for the BMV in HD-map. Store the similarity to BMV D_{BMV} ;
- 3) If $D_{BMV} < D_{min}$ then update the lowest similarity variable $D_{min} = D_{BMV}$. Store data hypervector as a candidate for the update.
- 4) Use the hypervector with lowest similarity for the update of hypervector \mathbf{s} (7) on the current iteration.

E. Finding the target node on HD-map

The hypervector found during the WMS procedure must be anchored to a hypervector \mathbf{p}_{target} , encoding a node on HD-map. Since the WMS procedure finds a hypervector, which unbinds a hypervector of HD-map with the lowest similarity, intuitively, it should be bound to a different hypervector. Using the FFTR heuristic, this new hypervector should be located further away from the current BMV in order to create a new point of attraction for the hypervector found by the WMS procedure and other hypervectors similar to it.

With the FPE encoding of HD-map's hypervectors the availability of farthest hypervectors to one selected hypervector is different depending on the choice of the bandwidth parameter for the same size of the map. For large values of $\epsilon_{\mathcal{P}}$ the similarity between hypervectors encoding neighboring nodes decays faster than for small values of $\epsilon_{\mathcal{P}}$, therefore, the number of dissimilar hypervectors is larger in the former case. This is demonstrated in Fig. 4a and 4b for $\epsilon_{\mathcal{P}} = 0.03$ and Fig. 4c and 4d for $\epsilon_{\mathcal{P}} = 0.008$ on 200×200 HD-map.

The simplest heuristic for finding the hypervector farthest to the given BMV for HD-maps with large $\epsilon_{\mathcal{P}}$ is, therefore, a random selection of \mathbf{p}_{target} . As we demonstrate in the next section, this heuristic leads to an adequate accuracy performance of Hyperseed in classification tasks. At the same time, obviously, the visualization of such projections is not

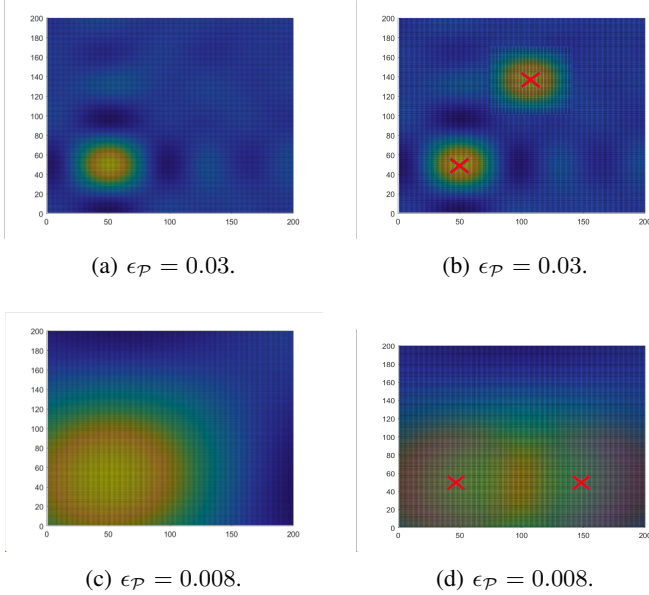


Fig. 4: Distribution of cosine similarities on HD-map for different values of FPE bandwidth $\epsilon_{\mathcal{P}}$.

very informative since it does not adequately display the internal disposition of classes.

In the case of small values of $\epsilon_{\mathcal{P}}$, the number of farthest hypervectors on HD-map is limited and has to be selected according to a certain heuristic. When Hyperseed is used in visualization tasks, $\epsilon_{\mathcal{P}}$ is chosen such that most dissimilar hypervectors are located in the corners of HD-map. These corner nodes are then chosen as $\mathbf{p}_{\text{target}}$ during the update phase.

Fig. 5 demonstrates an instance of projections of a dataset containing collection of n -gram statistics from texts on seven European languages. The dataset is projected onto a 20×20 HD-map with $\epsilon_{\mathcal{P}} = 0.008$. The complete experiment description follows in the next section. In the figure, crosses in the corners of HD-map show the choice of target nodes during the update procedure. We observe a semantically meaningful projections of languages. The color of the crosses corresponds to the class of the hypervector selected by the WMS procedure. One most important observation at this point is that the classes that were not used for the update of hypervectors, e.g., Swedish, French, and Bulgarian languages, emerged automatically and adequately projected.

F. The iterative Hyperseed algorithm

Fig. 6 displays all phases of the Hyperseed algorithm in a flowchart. The hyperparameters of the algorithm are dimensionality of hypervectors d and the number of iterations I (i.e., the number of updates of hypervectors). In the flowchart, function `SelectD()` returns an arbitrary hypervector from \mathcal{D} if its argument is “any” or j -th hypervector from \mathcal{D} when it is called with argument j . Function `SelectP()` returns a hypervector from \mathcal{D} as described in the previous subsection. Function `FindBMV($\mathbf{p}^*, \mathcal{P}$)` performs a search in the associative memory storing hypervectors of \mathcal{P} and returns the hypervector with the closest cosine similarity to \mathbf{p}^* . Function

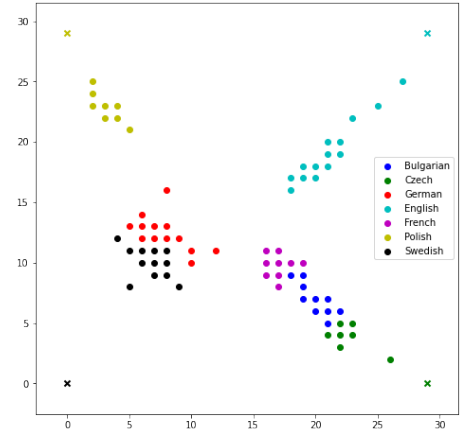


Fig. 5: Projection of seven European languages after four updates. The chosen $\mathbf{p}_{\text{target}}$ hypervectors are four nodes $(0,0)$, $(20,0)$, $(20,20)$ and $(0,20)$.

$\text{Sim}(\mathbf{p}^*, \mathbf{BMV})$ computes the cosine similarity between the two hypervectors.

The computational complexity of Hyperseed is $\mathcal{O}(INd|\mathcal{P}|)$, where I is the number of iterations (updates) of Hyperseed, N is the number of training data hypervectors, d is the dimensionality of hypervectors, and $|\mathcal{P}|$ is the size of HD-map. When implemented on neuromorphic hardware the search of the BMV happens in a constant time τ , therefore, the time complexity of Hyperseed in this case is $\mathcal{O}(IN) \cdot \tau$. The memory complexity, of course, depends on the size of HD-map which is $d \times |\mathcal{P}|$.

The complexity of the SOM algorithm in the Winner-Takes-All phase and in the weight matrix update procedures is $\mathcal{O}(INd|SOM|)$. Here, I is the number of iterations of SOM algorithm, N is the number of data samples, $|SOM|$ is the number of nodes in the SOM map (corresponds to $|\mathcal{P}|$ in Hyperseed) and d is the number of neurons per node (corresponds to the dimensionality of hypervectors in Hyperseed). While the \mathcal{O} complexities of the two algorithms are matching, Hyperseed uses a single vector operation in the update phase and substantially fewer number of iterations.

V. EXPERIMENTS AND RESULTS

This section describes the results of the experimental evaluation of the proposed Hyperseed algorithm. Before elaborating on the details of the experimental evaluation, it is important to set realistic expectations in order to correctly interpret the results. In particular, in the classification tasks, it would not be reasonable to expect a performance comparable to, e.g., deep learning-based models. This is because classification is not the primary task for unsupervised learning approaches. As any other unsupervised learning algorithm, Hyperseed does not modify the input representations to make them more separable. We chose to evaluate the performance of Hyperseed on classification tasks mainly because visualization (as one of the main applications of the unsupervised learning) is subjective and it is hard to quantify its quality.

We report three illustrative cases: 1) unsupervised learning from one-shot demonstrations on six synthetic datasets; 2)

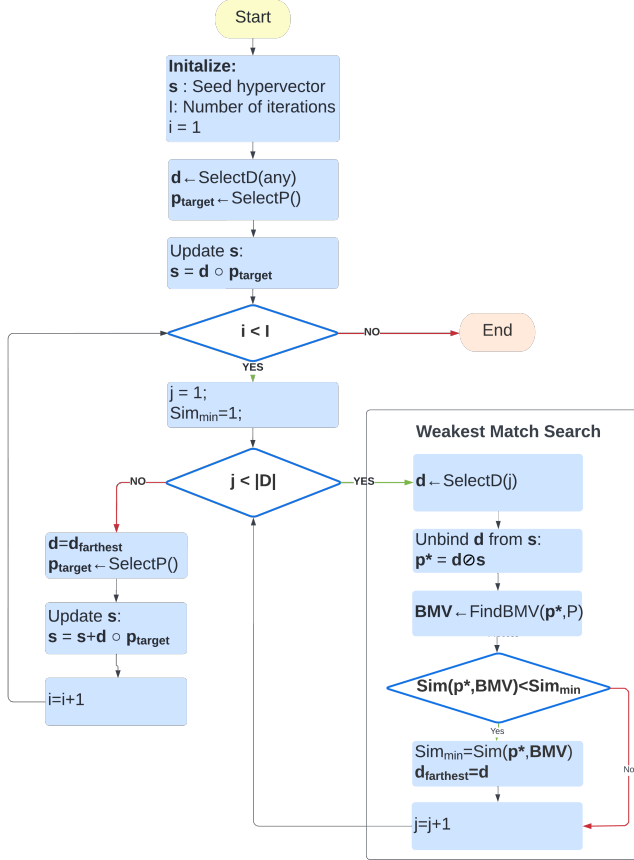


Fig. 6: Flowchart of Hyperseed.

classification of the Iris dataset; and 3) identification of 21 languages using their n -gram statistics. In all the experiments, we used dense complex FHRR representations [94] of varying dimensionality. The Python implementation of the Hyperseed algorithm and the code necessary to reproduce all the experiments reported in this study are publicly available⁴.

A. Data transformation to high-dimensional space

In the first two experiments, where input data are in the form of feature vectors of dimensionality K , the values of each feature $f_k, k \in [1, K]$ were normalized to range $[0, 1]$. The interval $[0, 1]$ was then split into q quantization levels. For each feature a base hypervector of unit length \mathbf{b}_k was randomly generated. Then levels for the particular feature \mathbf{I}_i^k were encoded in FHRR representation using FPE [19]–[23]: $\mathbf{I}_i^k = \mathbf{b}_k^{\epsilon_i}$, where $\epsilon \in \mathbb{R}$ represents the bandwidth parameter. The feature vector of a data sample was then represented as a single hypervector with the superposition operation: $\mathbf{v} = \sum_{k \in K} \mathbf{I}_i^k$.

In the language identification experiments, n -gram statistics was represented as hypervectors following the procedure in [68], [71]. First, a bijection of the alphabet letters $i \in |\mathcal{A}|$ to random unitary atomic hypervectors \mathbf{b}_i was created. To encode

the position of character i in the n -gram, the permutation operation was used on the corresponding atomic hypervector \mathbf{b}_i . For example, the hypervector for character “b” on a third position in a tri-gram is the corresponding atomic hypervector rotated three times: $\rho^3(\mathbf{b}_b)$. An n -gram of size n was encoded as binding of position based encoded hypervectors for corresponding characters. For example, a tri-gram “bdf” was encoded as: $\rho(\mathbf{b}_b) \circ \rho^2(\mathbf{b}_d) \circ \rho^3(\mathbf{b}_f)$. The n -gram statistics for a given text sample was then encoded into a single hypervector through the superposition of hypervectors for all observed n -grams.

B. Hyperseed for classification tasks

The proposed Hyperseed algorithm is by definition an unsupervised learning algorithm, therefore, an extra mechanism is needed to use it in supervised tasks such as the considered Iris classification and language identification task. Once Hyperseed was trained, there is a need to assign labels to the best matching hypervectors in HD-map.

Recall that Hyperseed is trained on a small subset of the available training data as described in Section IV-F. For the labeling process in the experiments presented below, the training data were presented to the trained Hyperseed HD-map for one full epoch that did not update the seed hypervector \mathbf{s} . The labels of the training data were used to calculate statistics for the BMV in HD-map. The nodes were assigned labels of the input samples that were prominent in the collected statistics.

At the classification phase, hypervectors of the nodes with the assigned labels were stored in the memory. During the classification phase, samples of the test data were used to assess the trained Hyperseed. For each sample in the test data, the BMV in HD-map was determined using the search procedure (Section IV-B). The test sample was then assigned the label of the closest labeled hypervector stored in the memory.

Accuracy was used as the main performance metric for evaluation and comparison of Hyperseed runs with different parameters. It should be re-emphasised that the focus of experiments was not on achieving the highest possible accuracy but on a comparative analysis of the Hyperseed algorithm.

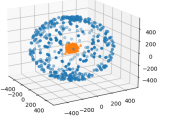
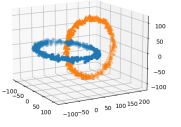
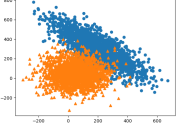
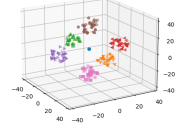
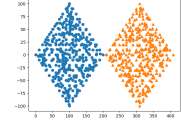
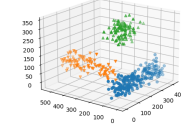
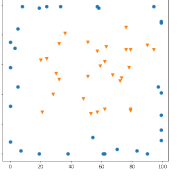
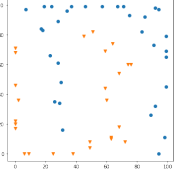
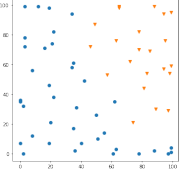
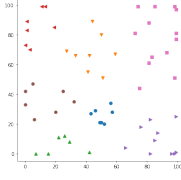
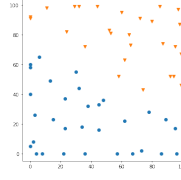
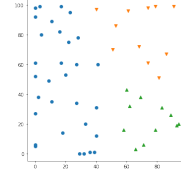
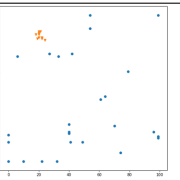
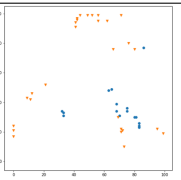
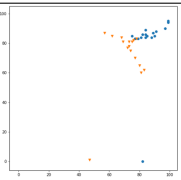
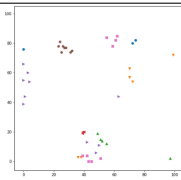
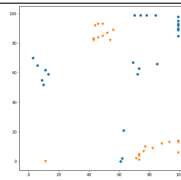
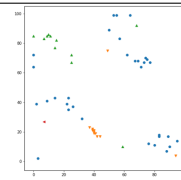
C. Experiment 1: The performance of Hyperseed on synthetic datasets with one-shot demonstration

This experiment serves the purpose of highlighting the major property of the Hyperseed algorithm – the capability of one-shot learning. When talking about one-shot learning, one has to be careful with its definition. In many practical cases, a single example is obviously not enough for accurate inference, instead, it is reasonable to talk about learning from a limited number of data samples, i.e., few-shot learning. This is what we intend to gradually demonstrate with all the experiments in this section.

In the first experiment, we fix the number of updates of seed hypervector \mathbf{s} to one. This, essentially, boils down to randomly picking a sample from the particular training data, running the Hyperseed’s update phase, and right after that performing

⁴Implementation of Hyperseed and the experiments, 2022. [Online.] Available: <https://github.com/eaoltu/hyperseed>

TABLE I: Comparison of projections and classification accuracy of Hyperseed vs. SOMs on FCPS datasets

Dataset	Atom	Chain link	Engy time	Hepta	Two diamonds	Lsun 3D
						
SOM						
	Accuracy: 0.8878	Accuracy: 0.9265	Accuracy: 0.9528	Accuracy: 0.9777	Accuracy: 0.9774	Accuracy: 0.9715
Hyperseed						
	Accuracy: 0.9821	Accuracy: 0.9780	Accuracy: 0.9071	Accuracy: 0.9552	Accuracy: 0.9902	Accuracy: 0.9005

labeling, classification on the corresponding test data, and the visualization.

For this purpose, we used synthetic datasets from Fundamental Clustering Problems Suite (FCPS) [95]. FCPS provides several non-linear but simple datasets that can be visualized in two or three dimensions, for elementary benchmarking of clustering and non-linear classification algorithms.

We selected six FCPS datasets that are most representative of the non-linearity, which we aim to learn using Hyperseed, they are: “Atom”, “Chain Link”, “Engy Time”, “Hepta”, “Two Diamonds”, and “Lsun 3D”. We repeated the experiment eight times. In each run, all hypervectors used for data encoding as well as for the operation of the Hyperseed algorithm were generated with a new seed used to initialize a pseudorandom generator.

In Table I, we present the results of this experiment in the form of the comparative evaluation of the projections by the conventional SOM and the projections produced by the Hyperseed algorithm, both visually and as the classification accuracy for each dataset. The sizes of the SOM grid and HD-map of the Hyperseed algorithm were the same 100×100 . The first row in the table presents the visualization of the six selected datasets in their original two or three dimensional data space. The second row presents HD-map projections and classification accuracy of the conventional SOM, while the third row shows HD-map projections and classification accuracy of the Hyperseed algorithm.

The primary observation in relation to the classification is that the Hyperseed algorithm provided average accuracy on a par with the conventional SOM (0.948 versus 0.943).

The visualization of HD-map projections of Hyperseed are more representative of the topology preservation of the original data space, in comparison to the conventional SOM. In three out of six datasets, Engy Time, Hepta, and Lsun 3D, the

topology preservation was directly comparable to the original dataset, where data samples of the same class were tightly clustered, in contrast to the conventional SOM, where these data samples were more scattered. For the remaining three datasets (Atom, Chain Link and Two Diamonds), although the topology preservation was not representative, the classification accuracy was high. This can be rationalized by the fixed 2D structure, which inhibits the complete visualization of the data space.

D. Experiment 2: Iris classification with Hyperseed

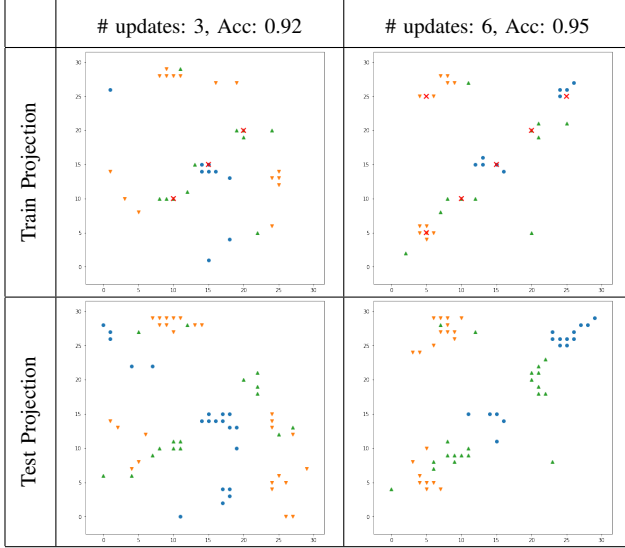
We continue the evaluation by exploring the details of Hyperseed’s operations during several updates. We used the Iris dataset with 150 samples of labeled data. The dataset contains three classes of Iris flowers described by four real-valued features. Data were encoded into hypervectors as described in Section V-A.

To further highlight the capabilities of Hyperseed to learn from few-shots, we decided to split the Iris dataset such that the size of the test data was larger than the size of the training data. The results for Hyperseed in this section were obtained for the 20%/80% split for training and test data, respectively. In order to provide a benchmark performance, we trained the conventional SOM with a 30×30 grid on different splits of the Iris dataset and counted the number of iterations it required the SOM to reach 95% accuracy. The experiments with the conventional SOM were repeated 10 times and the maximum accuracy across runs was recorded. Table II reports the results. One could see that for the 20/80 split the conventional SOM failed to achieve the target accuracy. Increasing the size of the training data allowed SOM reaching the target accuracy. The number of required iterations, however, was significant (the lowest was 200 for the 80/20 split). The maximum number of

TABLE II: Number of iterations of the conventional SOM required to reach 95% accuracy for different splits of Iris.

Data set split (training/testing), %	20/80	40/60	60/40	80/20
Number of updates	-	2500	600	200

TABLE III: Comparison of the Hyperseed projections with different random seeds for the Iris dataset.



iterations of Hyperseed was set to 3 and 6. This means that algorithm performed 90 (3 times 30 samples of the training data) and 180 (6 times 30 samples of the training data) searches for the BMV and only 3 (and correspondingly 6) updates of seed hypervector s . This has to be compared to 200×120 of searches and updates of the conventional SOM in the 80/20 split case. Each experiment (with three and six updates) was run ten times with different seeds.

The target nodes at each update were pre-selected to be (15,15) - for the first update, (20,20) - for the second update, (10,10) - for the third update, (5,5) - for the fourth update, (25,25) - for the fifth update, and (5,25) - for the sixth update. These nodes are marked by red crosses in the visualizations. This highlights again the importance of the target node selection rule for visually adequate projections. In this experiment, however, the adequate visualization was not important since our focus was on the performance characteristics of the Hyperseed algorithm in the classification task. Table III shows the results of the experiment. For the fair comparison with the conventional SOM here we also select the best performance across runs.

The first interesting observation comes in the case of three updates. We selected the best runs, which resulted in the highest accuracy on the test data (0.93). The projection of the training data show that out of three updates, in total one update was done for the first class (blue circles) and two updates were done for the second class. Thus, the cluster for the third class emerged automatically.

Another important observation comes from the relative placement of the data samples (both from the training and test

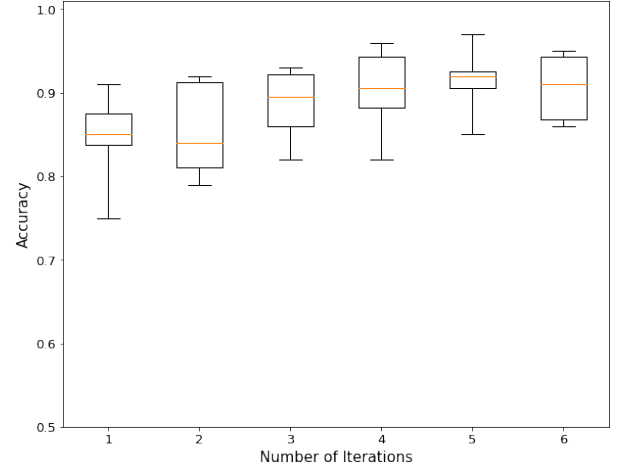


Fig. 7: Number of iterations against the accuracy ($d = 500$) of Hyperseed on the Iris dataset.

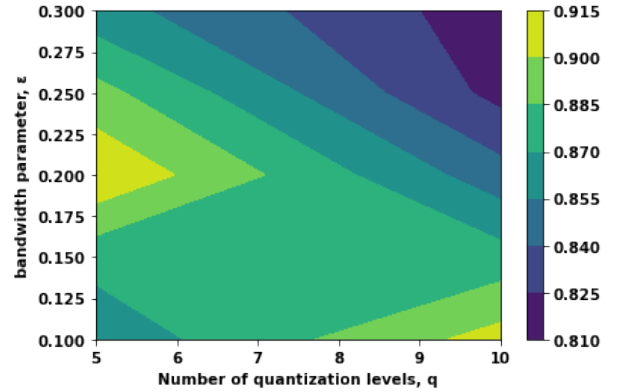
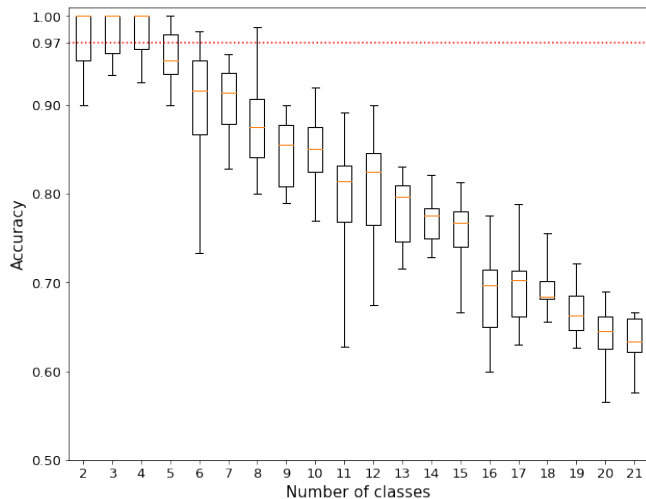
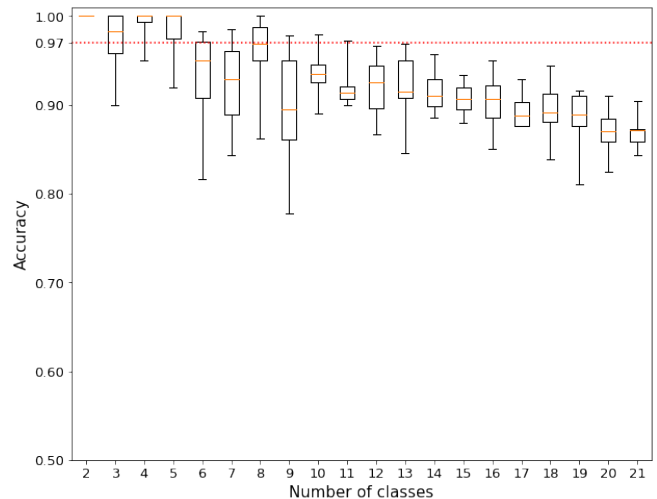


Fig. 8: Classification accuracy against the choice of FPE hyperparameters for data encoding: bandwidth and the number of quantization levels. Dimensionality of hypervectors was set to $d = 500$.

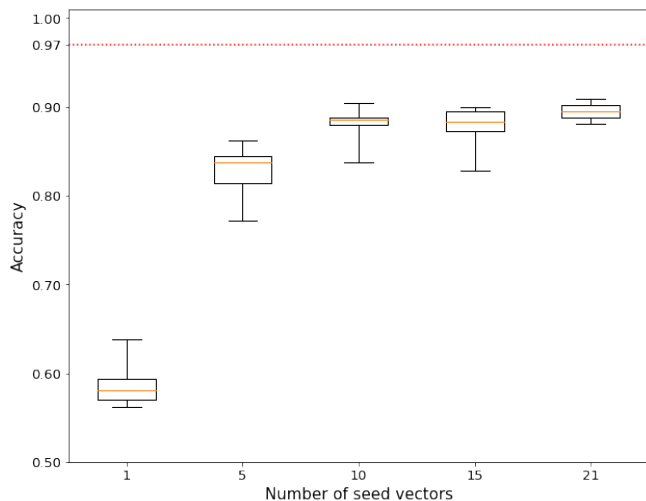
data) on HD-map. For the Iris dataset, it is known that the second and third classes are very similar to each other, which manifests in misclassification of some of their samples. Here, we see that this was, indeed, the case (orange triangles are very close to green triangles in several nodes of HD-map). However, in the case of Hyperseed this proximity did not lead to large degradation of the classification accuracy. This is because each point in HD-map attracted similar samples. Next, Fig. 7 shows the accuracy of Hyperseed when increasing the number of updates. On average, the classification accuracy increased with more updates of seed hypervector s . Also, in Fig. 8 we demonstrate the tradeoff between the choice of hyperparameters for the FPE encoding of input data (bandwidth ϵ and the number of quantization levels, q) and the classification accuracy. We observed that the best accuracy was achieved when ϵ and q were in the inverse relationship, that is when $\epsilon q = 1$.



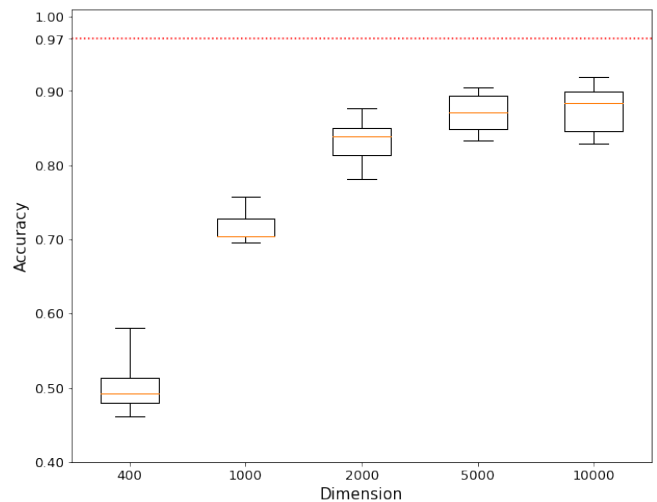
(a) Accuracy with one seed hypervector s using the original update rule from Section IV-C, $d = 5000$.



(b) Number of classes vs. accuracy with original and modified learning phases ($d = 5000$).



(c) Number of s hypervectors vs. accuracy ($d = 5000$).



(d) Dimensionality of hypervectors vs. accuracy (Number of s hypervectors is 10).

Fig. 9: The performance of Hyperseed in the task of language identification.

E. Experiment 3: Training Hyperseed on n -grams in language identification task

While the previous two experiments were used to get first insights on the major properties of Hyperseed, with this experiment we intend to demonstrate its performance on a larger scale problem. We will use it for the classification task of 21 European languages using n -gram statistics. The list of languages is as follows: Bulgarian, Czech, Danish, German, Greek, English, Estonian, Finnish, French, Hungarian, Italian, Latvian, Lithuanian, Dutch, Polish, Portuguese, Romanian, Slovak, Slovene, Spanish, Swedish. The training data is based on the Wortschatz Corpora [96]. The average size of each language corpus in the training data was 1085637.3 ± 121904.1 symbols. In this study, we use the method for encoding n -gram statistics into hypervectors from [71], where it was used as an input to the conventional SOM. Each original language corpus was divided into samples where the length of each sample

was set to 1000 symbols. The total number of samples in the training data was 22791.

The test data also contains samples from the same languages and is based on the Europarl Parallel Corpus⁵. The total number of samples in the test data was 21000, where each language was represented with 1000 samples. Each sample in the test data corresponded to a single sentence. The average size of a sample in the test data was 150.3 ± 89.5 symbols.

The data for each language was pre-processed such that the text included only lower case letters and spaces. All punctuation was removed. Lastly, all text used the 26-letter ISO basic Latin alphabet, i.e., the alphabet for both training and test data was the same and it included 27 symbols. For each text sample, the n -gram statistics transformed to hypervectors was obtained, which was then used as input when training or testing Hyperseed. Since each sample was pre-

⁵Available online at <http://www.statmt.org/europarl/>.

processed to use the alphabet of only $a = 27$ symbols, the conventional n -gram statistics input was 27^n dimensional. In the experiment, we used tri-grams, therefore, the conventional representation was 19,683 dimensional. The dimensionality of the mapped n -gram statistics into hypervectors as described in Section V-A depends on the dimensionality of the hypervectors d . The results reported below were obtained for different dimensionalities in the range [400, 10000], which corresponds to the dimensionality reduction of the original representation space from 49 fold (for $d = 400$) to 2 fold (for $d = 10,000$). In this experiment, we used 100×100 HD-map.

The first investigation was performed with dimensionality of hypervectors $d = 5000$ (4 fold dimensionality reduction). In this experiment, we exposed Hyperseed to a different number of classes from the original dataset. The number of Hyperseed updates in each case was set relative to the number of classes at hand. Importantly, this heuristic for choosing the number of updates was adopted for automating experiments only. It only reflects the desire of keeping the number of iterations low. In the unsupervised learning context, the information about the number of classes is, obviously, unavailable.

The experiment was repeated eight times for each number of classes, each time selecting a different subset of languages for Hyperseed training. Fig. 9a shows the results. The reference accuracy 0.97 (the red line) is the accuracy obtained with the conventional tri-gram statistics representation reported in [68], [97] using the nearest neighbor classifier. The main observation to make from this investigation is that the performance of Hyperseed dropped as it was exposed to larger number of classes. The explanation to this is connected to the finite capacity of hypervectors in terms of the number of hypervectors one can superimpose together while keeping the sufficient accuracy of retrieving them back [98], [99]. As we discussed above, each update to seed hypervector s introduces noise to the previous updates. That is why the number of updates in Hyperseed needs to be kept low.

1) *Modified Hyperseed learning phase:* To mitigate the problem of the reduced accuracy on large problems (in terms of the number of distinct classes) the learning phase of Hyperseed was modified.

Instead of having a single seed hypervector, it is proposed to use N vectors $s_i, i = 1..N$, where N becomes another hyperparameter of Hyperseed. During the update procedure, these hypervectors will be updated in a round-robin manner in order to keep the balanced number of updates per a seed hypervector.

During the search of the BMV in either the WMS or test procedures, the binding in (1) is now computed for the current input hypervector and all seed hypervectors s_i 's. Hypervector p with the highest cosine similarity across all results of unbinding with all s_i is selected as the BMV.

The number of iterations (and as the result the number of updates) with the new update phase should be scaled such that the number of updates per s_i is approximately the same. For example, in the experiments with $N = 10$ the number of iterations was configured to 30 to allow for three updates per s_i . Fig. 9b demonstrates significant performance improvement

with the modified learning phase. The maximum performance of Hyperseed in 21 classes case is 0.91 after 30 iterations.

Next, we are interested in the effect of the number of seed hypervectors on the classification performance of Hyperseed. Fig. 9c shows the classification accuracy in the case of 21 classes for different number of seed hypervectors used in the modified learning phase. The main observation here was that the performance stabilized after a certain value of this parameter. In our case, there was no significant increase in the accuracy after using ten seed hypervectors.

2) *Hyperseed performance for different dimensionalities of hypervectors:* Finally, we are interested in the effect of the dimensionality of hypervectors on the performance of Hyperseed. We performed an experiment with ten seed hypervectors and varied the dimensionality of the hypervectors used by the algorithm from 400 until 10000. The results are depicted in Fig. 9d. The main observation to make was that the classification accuracy on small dimensionalities was low, as expected. On the positive side, it was substantially higher than the random choice, which is 0.04 in this case. The reason for this is rather clear and it is connected to the dimensionality of the non-distributed representations, which is high (19,683). With only 400 dimensions and the adopted encoding procedure of the input data, we operated well above the capacity of the hypervectors, this lead to high inter-class similarity. Increasing the dimensionality allowed addressing this issue and resulted in better accuracy. It turned out that in the case of the language identification 5000 was the optimal dimensionality for obtaining high-quality performance. It is worth noting that the accuracy of Hyperseed was still lower by 0.07 compared to the baseline. It was, however, not expected that it would necessarily achieve higher accuracy compared to the supervised methods.

VI. DISCUSSION

Having presented the Hyperseed algorithm and its empirical evaluation, it is now pertinent to discuss the following aspects that require further investigation, Hyperseed on neuromorphic hardware, performance comparison of embeddings and limitations of Hyperseed.

A. Hyperseed on neuromorphic hardware

The Hyperseed algorithm from the start was designed targeting an implementation on the neuromorphic hardware. This target departs from recent developments within the Intel neuromorphic research community, where VSA is promoted as an algebraic framework for the development of algorithms on Intel's Loihi [12], [64], [100].

Since the main focus of this article is on the algorithmic aspects of Hyperseed, we resort to making rather high level links to a neuromorphic realization of algorithm's operations and present an evaluation of its computational bottleneck using the existing neuromorphic implementation.

Both HRR and FHRR representations could be mapped onto activities of spiking neurons. In the case of FHRR, the phase of components is used for phase-to-spike-timing mapping. In this way, FHRR representation does not result in higher memory

footprint as in the case of the CPU implementation. VSA operations are realized in either Resonate-and-Fire neurons [64] or in Leaky Integrate and Fire neurons [65], [101].

In the case of HRR, real-valued components are used in spike-time latency code, where earlier spikes represent larger magnitudes. VSA operations can be realized by Leaky Integrate and Fire neurons. In this article, we use the realization of the dot product calculation presented in [100] (also based on Leaky Integrate and Fire neurons) in order to demonstrate the feasibility of neuromorphic implementation of computational bottleneck of Hyperseed – the search for the BMV in HD-map for an unbound noisy hypervector \mathbf{p}^* resulting from (6).

To measure the performance of the Hyperseed’s search procedure in the language identification task HD-maps of various sizes (starting from 30×30 and incrementing the grid size by 10 along each axis until 90×90) were generated as in Section IV-A. The hypervectors of each HD-map were then used for mapping their values onto spiking activity of the k NN reference base on Intel’s Loihi-based Nahuku-32 neuromorphic system. This operation was performed only once as part of the initialization since HD-map remains unchanged for the life-time of Hyperseed. Therefore, the time to construct the k NN reference base was not taken into account in the run-time performance evaluation.

For the experiment we chose the case of identifying five randomly selected languages with a single seed hypervector \mathbf{s} as the reference scenario. The original dimensionality of the hypervectors used for the encoding of the input data as well as for all other hypervectors of the Hyperseed algorithm was $d = 10000$. The reference scenario accuracy of Hyperseed obtained on a CPU was 0.84.

k NN on Loihi was used to model the search operation during the labeling and testing process. To do this, seed hypervector \mathbf{s} was pre-trained offline on the CPU as described in Section IV-C. For the labeling process, the binding of all training and test data with the trained \mathbf{s} was performed and the results (the noisy versions of the BMVs) were used as queries to the k NN reference base storing HD-map on Loihi. The dimensionality of the existing Loihi implementation of k NN is $d_{kNN} < 512$, which is a platform specific limit. Note, that in VSA the dimensionality of hypervectors is connected to the information capacity of the superposition. In Hyperseed, every update of seed hypervector \mathbf{s} increases the cross-talk noise to previous bindings $\mathbf{s} \circ \mathbf{p}_{\text{target}}$. Fig. 9a demonstrated the accuracy degradation for dimensionality $d = 5000$ with the increase of the number of updates of \mathbf{s} . For smaller dimensionalities, the number of updates for maintaining the acceptable accuracy is even lower. Therefore, in this experiment the training of Hyperseed was done on higher dimensionalities and then the dimensionality was reduced using principal component analysis to $d = 400$ to meet the current limitations of the existing implementation.

To label BMVs, the training data (after binding with \mathbf{s}) were used as queries to the k NN reference base and the top-1 index for each query (the earliest fired output neuron) was recorded. After that, the labeling was performed on the CPU using the procedure described above. To compute the accuracy, the test data hypervectors (also after binding with \mathbf{s}) were used

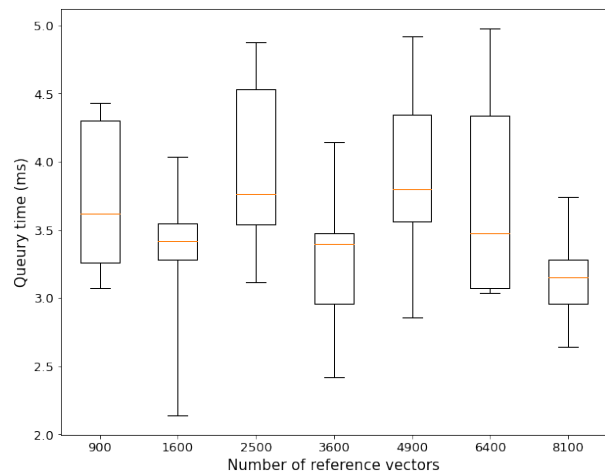


Fig. 10: Time of querying a noisy BMV in HD-map implemented as the k NN reference base against the size of HD-map in the number of reference hypervectors.

as queries to the k NN reference base. The top-1 index was recorded and used to compute the accuracy against the list of the labeled indices.

The average measured accuracy on Loihi was 0.84, which matched the reference accuracy on the CPU implementation of Hyperseed. This means that the neuromorphic implementation of HD-map and the calculation of similarity did not introduce sensible errors.

Next, in addition to the accuracy we also measured the query time to the k NN reference base for different sizes of HD-map. The main outcome of this experiment is illustrated by Fig. 10. It shows the computational benefit of implementing the bottleneck operation of Hyperseed on in the neuromorphic hardware: due to parallel, power-efficient computation of the dot product in Loihi, Hyperseed, as expected, was empowered with constant-time search for different sizes of HD-map.

B. Performance comparison of embeddings and distributed representations

While a large body of knowledge on methods for encoding data into hypervectors is accumulated throughout the years, the problem is still considered as of the primary importance in the area of VSA. Hyperseed in this respect offers a playground for comparing different embeddings as the embeddings of high quality lead to higher accuracy in classification tasks. It is particularly important to develop the operations of Hyperseed on sparse representations.

C. Limitations of Hyperseed and future developments

Hyperseed when using 2D HD-map as evaluated in this article is limited in its capability to describe complex manifold structures. In this sense the algorithm does not show advantages over Self-Organizing Maps, which have similar limitations. This case was chosen to demonstrate the feasibility of implementing non-trivial learning functionality with straight-forward VSA operations, which in itself is an original research

contribution. However, Hyperseed is in fact scalable in terms of two other aspects: 1) topologies of higher dimensionalities than two as it does not require updates of the hypervectors in HD-map and 2) topologies of other structures than regular grid due to the generality of FPE encoding. The investigation of this capability is part of an ongoing work on Hyperseed extension.

VII. CONCLUSIONS

The increasing accumulation of unstructured and unlabelled big data initiated a renewed interest in unsupervised machine learning algorithms that are able to capitalize on computational efficiencies of biologically-inspired neuromorphic hardware. In this article, we presented the Hyperseed algorithm that addresses these challenges through the manifestation of a novel unsupervised learning approach that leverages Vector Symbolic Architectures for fast learning from only few input vectors and single vector operation learning rule implementation. A further novelty is that it implements the entire learning pipeline purely in terms of operations of Vector Symbolic Architectures. Hyperseed has been empirically evaluated across diverse scenarios: synthetic datasets from Fundamental Clustering Problems Suite, benchmark classification using the Iris dataset, and the more practical classification of 21 European languages using their n -gram statistics. As future work, we will work on the adaptation of the Hyperseed algorithm for neuromorphic hardware.

REFERENCES

- [1] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2019.
- [2] B. Cheung, A. Terekhov, Y. Chen, P. Agrawal, and B. Olshausen, "Superposition of Many Models into One," in *Advances in Neural Information Processing Systems (NIPS)*, 2019, pp. 10868–10877.
- [3] P. Neubert, S. Schubert, and P. Protzel, "An Introduction to Hyperdimensional Computing for Robotics," *KI - Künstliche Intelligenz*, vol. 33, no. 4, pp. 319–330, 2019.
- [4] M. Hersche, P. Rupp, L. Benini, and A. Rahimi, "Compressing Subject-specific Brain-Computer Interface Models into One Model by Superposition in Hyperdimensional Space," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 246–251.
- [5] D. Kleyko, M. Kheffache, E. P. Frady, U. Wiklund, and E. Osipov, "Density Encoding Enables Resource-Efficient Randomly Connected Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3777–3783, 2021.
- [6] P. Neubert and S. Schubert, "Hyperdimensional Computing as a Framework for Systematic Aggregation of Image Descriptors," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 16938–16947.
- [7] C.-Y. Chang, Y.-C. Chuang, and A.-Y. A. Wu, "Task-Projected Hyperdimensional Computing for Multi-task Learning," in *IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI)*, ser. IFIP Advances in Information and Communication Technology, vol. 583, 2020, pp. 241–251.
- [8] —, "IP-HDC: Information-Preserved Hyperdimensional Computing for Multi-Task Learning," in *IEEE Workshop on Signal Processing Systems (SiPS)*, 2020, pp. 1–6.
- [9] G. Karunaratne, M. L. Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-Memory Hyperdimensional Computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.
- [10] G. Karunaratne, M. Schmuck, M. L. Gallo, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, "Robust High-dimensional Memory-augmented Neural Networks," *Nature Communications*, vol. 12, no. 1, pp. 1–12, 2021.
- [11] D. Kleyko, G. Karunaratne, J. M. Rabaey, A. Sebastian, and A. Rahimi, "Generalized Key-Value Memory to Flexibly Adjust Redundancy in Memory-Augmented Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 99, no. PP, pp. 1–6, 2022.
- [12] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataraman, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [13] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional Computing as a Nanoscalable Paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017.
- [14] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, and F. T. Sommer, "Vector Symbolic Architectures as a Computing Framework for Emerging Hardware," *Proceedings of the IEEE*, pp. 1–34, 2021.
- [15] A. N. Gorban and I. Y. Tyukin, "Blessing of Dimensionality: Mathematical Foundations of the Statistical Physics of Data," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 376, no. 2118, pp. 1–18, 2018.
- [16] 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.
- [17] T. Kohonen, *Self-Organizing Maps*. Springer Series in Information Sciences, 2001.
- [18] D. Kleyko, E. Osipov, D. D. Silva, U. Wiklund, and D. Alahakoon, "Integer Self-Organizing Maps for Digital Hardware," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.
- [19] T. A. Plate, *Distributed Representations and Nested Compositional Structure*. University of Toronto, PhD Thesis, 1994.
- [20] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on Functions Using Randomized Vector Representations," *arXiv:2109.03429*, pp. 1–33, 2021.
- [21] —, "Computing on Functions Using Randomized Vector Representations (in brief)," in *Neuro-Inspired Computational Elements Conference (NICE)*, 2022, pp. 115–122.
- [22] B. Komer, T. C. Stewart, A. R. Voelker, and C. Eliasmith, "A Neural Representation of Continuous Space using Fractional Binding," in *Annual Meeting of the Cognitive Science Society (CogSci)*, 2019, pp. 2038–2043.
- [23] B. Komer, *Biologically Inspired Spatial Representation*. University of Waterloo, PhD Thesis, 2020.
- [24] D. A. Rachkovskij, "Representation and Processing of Structures with Binary Sparse Distributed Codes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 2, pp. 261–276, 2001.
- [25] D. Kleyko, E. Osipov, and D. A. Rachkovskij, "Modification of Holographic Graph Neuron using Sparse Distributed Representations," *Procedia Computer Science*, vol. 88, pp. 39–45, 2016.
- [26] 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*, vol. PP, no. 99, pp. 1–14, 2021.
- [27] C. Eliasmith, *How to Build a Brain*. Oxford University Press, 2013.
- [28] D. A. Rachkovskij, "Some Approaches to Analogical Mapping with Structure Sensitive Distributed Representations," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 16, no. 3, pp. 125–145, 2004.
- [29] D. A. Rachkovskij and S. V. Slipchenko, "Similarity-based Retrieval with Structure-Sensitive Sparse Binary Distributed Representations," *Computational Intelligence*, vol. 28, no. 1, pp. 106–129, 2012.
- [30] D. Kleyko, E. Osipov, and R. Gayler, "Recognizing Permuted Words with Vector Symbolic Architectures: A Cambridge Test for Machines," *Procedia Computer Science*, vol. 88, pp. 169–175, 2016.
- [31] M. N. Jones and D. J. K. Mewhort, "Representing Word Meaning and Order Information in a Composite Holographic Lexicon," *Psychological Review*, vol. 114, no. 1, pp. 1–37, 2007.
- [32] G. Recchia, M. Sahlgren, P. Kanerva, and M. N. Jones, "Encoding Sequential Information in Semantic Space Models: Comparing Holographic Reduced Representation and Random Permutation," *Computational Intelligence and Neuroscience*, pp. 1–18, 2015.
- [33] D. A. Rachkovskij and D. Kleyko, "Recursive Binding for Similarity-Preserving Hypervector Representations of Sequences," in *International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

- [34] D. A. Rachkovskij, "Shift-Equivariant Similarity-Preserving Hypervector Representations of Sequences," *arXiv:2112.15475*, pp. 1–10, 2021.
- [35] P. Jakimovski, H. R. Schmidtk, S. Sigg, L. W. F. Chaves, and M. Beigl, "Collective Communication for Dense Sensing Environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 4, no. 2, pp. 123–134, 2012.
- [36] D. Kleyko, N. Lyamin, E. Osipov, and L. Riliskis, "Dependable MAC Layer Architecture based on Holographic Data Representation using Hyper-Dimensional Binary Spatter Codes," in *Multiple Access Communications (MACOM)*, ser. Lecture Notes in Computer Science, vol. 7642, 2012, pp. 134–145.
- [37] H.-S. Kim, "HDM: Hyper-Dimensional Modulation for Robust Low-Power Communications," in *IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.
- [38] D. Kleyko, E. Osipov, and U. Wiklund, "A Hyperdimensional Computing Framework for Analysis of Cardiorespiratory Synchronization During Paced Deep Breathing," *IEEE Access*, vol. 7, pp. 34 403–34 415, 2019.
- [39] E. Osipov, D. Kleyko, and A. Legalov, "Associative Synthesis of Finite State Automata Model of a Controlled Object with Hyperdimensional Computing," in *Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2017, pp. 3276–3281.
- [40] D. Kleyko, A. Rahimi, R. W. Gayler, and E. Osipov, "Autoscaling Bloom Filter: Controlling Trade-off Between True and False Positives," *Neural Computing and Applications*, vol. 32, pp. 3675–3684, 2020.
- [41] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey, "Classification and Recall with Binary Hyperdimensional Computing: Tradeoffs in Choice of Density and Mapping Characteristic," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5880–5898, 2018.
- [42] V. Christopher, T. Aathman, K. Mahendrakumar, R. Nawaratne, D. De Silva, V. Nanayakkara, and D. Alahakoon, "Minority Resampling Boosted Unsupervised Learning with Hyperdimensional Computing for Threat Detection at the Edge of Internet of Things," *IEEE Access*, 2021.
- [43] H. Moraliyage, S. Kahawala, D. D. Silva, and D. Alahakoon, "Evaluating the Adversarial Robustness of Text Classifiers in Hyperdimensional Computing," in *International Conference on Human System Interaction (HSI)*, 2022, pp. 1–8.
- [44] O. Rasanen and S. Kakouros, "Modeling Dependencies in Multiple Parallel Data Streams with Hyperdimensional Computing," *IEEE Signal Processing Letters*, vol. 21, no. 7, pp. 899–903, 2014.
- [45] A. Goltsev and D. A. Rachkovskij, "Combination of the Assembly Neural Network with a Perceptron for Recognition of Handwritten Digits Arranged in Numerical Strings," *Pattern Recognition*, vol. 38, no. 3, pp. 315–322, 2005.
- [46] D. A. Rachkovskij, "Representation of spatial objects by shift-equivariant similarity-preserving hypervectors," *Neural Computing and Applications*, pp. 1–17, 2022.
- [47] P.-C. Huang, D. Kleyko, J. M. Rabaey, B. A. Olshausen, and P. Kanerva, "Computing with Hypervectors for Efficient Speaker Identification," *arXiv:2208.13285*, pp. 1–5, 2022.
- [48] E. M. Kussul, L. M. Kasatkina, D. A. Rachkovskij, and D. C. Wunsch, "Application of Random Threshold Neural Networks for Diagnostics of Micro Machine Tool Condition," in *International Joint Conference on Neural Networks (IJCNN)*, vol. 1, 1998, pp. 241–244.
- [49] D. Kleyko, E. Osipov, N. Papakonstantinou, and V. Vyatkin, "Hyperdimensional Computing in Industrial Systems: The Use-Case of Distributed Fault Isolation in a Power Plant," *IEEE Access*, vol. 6, pp. 30 766–30 777, 2018.
- [50] M. Eggmann, A. Rahimi, and L. Benini, "A 5 μ W Standard Cell Memory-based Configurable Hyperdimensional Computing Accelerator for Always-on Smart Sensing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4116–4128, 2021.
- [51] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing, "HDCluster: An Accurate Clustering Using Brain-Inspired High-Dimensional Computing," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1591–1594.
- [52] T. Bandaragoda, D. D. Silva, D. Kleyko, E. Osipov, U. Wiklund, and D. Alahakoon, "Trajectory Clustering of Road Traffic in Urban Environments using Incremental Machine Learning in Combination with Hyperdimensional Computing," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 1664–1670.
- [53] A. Hernández-Cano, Y. Kim, and M. Imani, "A Framework for Efficient and Binary Clustering in High-Dimensional Space," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1859–1864.
- [54] M. Imani, S. Bosch, M. Javaheripi, B. D. Rouhani, X. Wu, F. Koushanfar, and T. Rosing, "SemiHD: Semi-Supervised Learning Using Hyperdimensional Computing," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.
- [55] M. Imani, Y. Kim, S. Riazzi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A Framework for Collaborative Learning in Secure High-Dimensional Space," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2019, pp. 435–446.
- [56] B. Khaleghi, M. Imani, and T. Rosing, "Prive-HD: Privacy-Preserved Hyperdimensional Computing," in *ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [57] A. Rosato, M. Panella, and D. Kleyko, "Hyperdimensional Computing for Efficient Distributed Classification with Randomized Neural Networks," in *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–10.
- [58] C.-Y. Hsieh, Y.-C. Chuang, and A.-Y. A. Wu, "FL-HDC: Hyperdimensional Computing Design for the Application of Federated Learning," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–5.
- [59] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations," *ACM Computing Surveys*, 2022.
- [60] —, "A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part II: Applications, Cognitive Models, and Challenges," *ACM Computing Surveys*, 2022.
- [61] G. Hinton, J. McClelland, and D. Rumelhart, "Distributed Representations," in *Parallel Distributed Processing. Explorations in the Microstructure of Cognition. Volume 1. Foundations*, D. Rumelhart and J. McClelland, Eds. MIT Press, 1986, pp. 77–109.
- [62] P. Kanerva, "A Family of Binary Spatter Codes," in *International Conference on Artificial Neural Networks (ICANN)*, 1995, pp. 517–522.
- [63] R. W. Gayler, "Multiplicative Binding, Representation Operators & Analogy," in *Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences*, 1998, pp. 1–4.
- [64] 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. 18 050–18 059, 2019.
- [65] A. Renner, Y. Sandamirskaya, F. T. Sommer, and E. P. Frady, "Sparse Vector Binding on Spiking Neuromorphic Hardware Using Synaptic Delays," in *International Conference on Neuromorphic Systems (ICONS)*, 2022, pp. 1–5.
- [66] G. Bent, C. Simpkin, Y. Li, and A. Preece, "Hyperdimensional Computing using Time-to-spike Neuromorphic Circuits," in *International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.
- [67] D. A. Rachkovskij, "Linear Classifiers based on Binary Distributed Representations," *Information Theories and Applications*, vol. 14, no. 3, pp. 270–274, 2007.
- [68] A. Joshi, J. Halseth, and P. Kanerva, "Language Geometry Using Random Indexing," in *Quantum Interaction (QI)*, 2016, pp. 265–274.
- [69] P. Alonso, K. Shridhar, D. Kleyko, E. Osipov, and M. Liwicki, "HyperEmbed: Tradeoffs Between Resources and Performance in NLP Tasks with Hyperdimensional Computing enabled Embedding of n-gram Statistics," in *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–9.
- [70] F. Mirus, P. Blouw, T. C. Stewart, and J. Conradt, "An investigation of vehicle behavior prediction using a vector power representation to encode spatial positions of multiple objects and neural networks," *Frontiers in Neurobotics*, vol. 13, pp. 1–17, 2019.
- [71] D. Kleyko, E. Osipov, D. D. Silva, U. Wiklund, V. Vyatkin, and D. Alahakoon, "Distributed Representation of n-gram Statistics for Boosting Self-Organizing Maps with Hyperdimensional Computing," in *International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI)*, ser. Lecture Notes in Computer Science, vol. 11964, 2019, pp. 64–79.
- [72] F. Mirus, T. C. Stewart, and J. Conradt, "The Importance of Balanced Data Sets: Analyzing a Vehicle Trajectory Prediction Model based on Neural Networks and Distributed Representations," in *International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.
- [73] K. Shridhar, H. Jain, A. Agarwal, and D. Kleyko, "End to End Binarized Neural Networks for Text Classification," in *Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, 2020, pp. 29–34.
- [74] E. M. Kussul, D. A. Rachkovskij, and D. C. Wunsch, "The Random Subspace Coarse Coding Scheme for Real-valued Vectors," in *Inter-*

- national Joint Conference on Neural Networks (IJCNN)*, vol. 1, 1999, pp. 450–455.
- [75] D. A. Rachkovskij, “Similarity-Reflecting Binary Vectors with Random Binary Projections,” *Cybernetics and Systems Analysis*, vol. 51, pp. 313–323, 2015.
- [76] A. Mitrokhin, P. Sutor, C. Fermuller, and Y. Aloimonos, “Learning Sensorimotor Control with Neuromorphic Sensors: Toward Hyperdimensional Active Perception,” *Science Robotics*, vol. 4, no. 30, pp. 1–10, 2019.
- [77] D. Kleyko, R. W. Gayler, and E. Osipov, “Commentaries on “Learning Sensorimotor Control with Neuromorphic Sensors: Toward Hyperdimensional Active Perception” [Science Robotics Vol. 4 Issue 30 (2019) 1-10],” *arXiv:2003.11458*, pp. 1–10, 2020.
- [78] M. Hersche, E. M. Rella, A. D. Mauro, L. Benini, and A. Rahimi, “Integrating Event-based Dynamic Vision Sensors with Sparse Hyperdimensional Computing: A Low-power Accelerator with Online Learning Capability,” in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2020, pp. 169–174.
- [79] D. Kleyko, E. P. Frady, M. Kheffache, and E. Osipov, “Integer Echo State Networks: Efficient Reservoir Computing for Digital Hardware,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1688–1701, 2022.
- [80] C. Diao, D. Kleyko, J. M. Rabaey, and B. A. Olshausen, “Generalized Learning Vector Quantization for Classification in Randomized Neural Networks and Hyperdimensional Computing,” in *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–9.
- [81] B. Igel'nik and Y. H. Pao, “Stochastic Choice of Basis Functions in Adaptive Function Approximation and the Functional-Link Net,” *IEEE Transactions on Neural Networks*, vol. 6, pp. 1320–1329, 1995.
- [82] M. Lukosevicius and H. Jaeger, “Reservoir Computing Approaches to Recurrent Neural Network Training,” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.
- [83] D. Kleyko, E. P. Frady, and F. T. Sommer, “Cellular Automata Can Reduce Memory Requirements of Collective-State Computing,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 6, pp. 2701–2713, 2022.
- [84] A. R. Voelker, P. Blouw, X. Choo, N. S.-Y. Dumont, T. C. Stewart, and C. Eliasmith, “Simulating and Predicting Dynamical Systems with Spatial Semantic Pointers,” *Neural Computation*, vol. 33, no. 8, pp. 2033–2067, 2021.
- [85] D. Kleyko, C. Bybee, C. J. Kymn, B. A. Olshausen, A. Khosrowshahi, D. E. Nikonov, F. T. Sommer, and E. P. Frady, “Integer Factorization with Compositional Distributed Representations,” in *Neuro-Inspired Computational Elements Conference (NICE)*, 2022, pp. 73–80.
- [86] K. Schlegel, P. Neubert, and P. Protzel, “HDC-MiniROCKET: Explicit Time Encoding in Time Series Classification with Hyperdimensional Computing,” in *International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.
- [87] A. A. Frolov, D. A. Rachkovskij, and D. Husek, “On Informational Characteristics of Willshaw-like Auto-associative Memory,” *Neural Network World*, vol. 12, no. 2, pp. 141–157, 2002.
- [88] A. A. Frolov, D. Husek, and D. A. Rachkovskij, “Time of Searching for Similar Binary Vectors in Associative Memory,” *Cybernetics and Systems Analysis*, vol. 42, no. 5, pp. 615–623, 2006.
- [89] V. I. Gritsenko, D. A. Rachkovskij, A. A. Frolov, R. W. Gayler, D. Kleyko, and E. Osipov, “Neural Distributed Autoassociative Memories: A Survey,” *Cybernetics and Computer Engineering*, vol. 2, no. 188, pp. 5–35, 2017.
- [90] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II, “An Analysis of Several Heuristics for the Traveling Salesman Problem,” *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977.
- [91] D. S. Hochbaum and D. B. Shmoys, “A Best Possible Heuristic for the k-Center Problem,” *Mathematics of Operations Research*, vol. 10, no. 2, pp. 180–184, 1985.
- [92] D. A. Rachkovskij, “Distance-Based Index Structures for Fast Similarity Search,” *Cybernetics and Systems Analysis*, vol. 53, pp. 636–658, 2017.
- [93] V. Chaudhary, R. Bhatia, and A. Ahlawat, “A novel Self-Organizing Map (SOM) learning algorithm with nearest and farthest neurons,” *Alexandria Engineering Journal*, vol. 53, 10 2014.
- [94] T. A. Plate, “Holographic Reduced Representations,” *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 623–641, 1995.
- [95] A. Ultsch, “Clustering with SOM: U* C,” in *Workshop on Self-Organizing Maps*, 2005.
- [96] U. Quasto, M. Richter, and C. Biemann, “Corpus Portal for Search in Monolingual Corpora,” in *International Conference on Language Resources and Evaluation (LREC)*, 2006, pp. 1799–1802.
- [97] A. Rahimi, P. Kanerva, and J. Rabaey, “A Robust and Energy Efficient Classifier Using Brain-Inspired Hyperdimensional Computing,” in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2016, pp. 64–69.
- [98] 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, pp. 1449–1513, 2018.
- [99] D. Kleyko, A. Rosato, E. P. Frady, M. Panella, and F. T. Sommer, “Perceptron Theory for Predicting the Accuracy of Neural Networks,” *arXiv:2012.07881*, pp. 1–12, 2020.
- [100] E. P. Frady, G. Orchard, D. Florey, N. Imam, R. Liu, J. Mishra, J. Tse, A. Wild, F. T. Sommer, and M. Davies, “Neuromorphic Nearest Neighbor Search Using Intel’s Pohoiki Springs,” in *Neuro-Inspired Computational Elements Workshop (NICE)*, 2020.
- [101] A. Renner, L. Supic, A. Danielescu, G. Indiveri, B. A. Olshausen, Y. Sandamirskaya, F. T. Sommer, and E. P. Frady, “Neuromorphic Visual Scene Understanding with Resonator Networks,” *arXiv:2208.12880*, pp. 1–15, 2022.



Evgeny Osipov received the Ph.D. degree in computer science from the University of Basel, Switzerland, in 2005. He is currently a Full Professor in dependable communication and computation systems with the Department of Computer Science and Electrical Engineering, Luleå University of Technology, Luleå, Sweden.

His research interests are in novel computational models for unconventional computer architectures. His current research focuses on hyperdimensional computing and vector symbolic architectures.



Sachin Kahawala received the B.S. degree (Hons.) in Computer Science from the University of Moratuwa, Sri Lanka. He is currently pursuing a Ph.D. in Artificial Intelligence in the Centre for Data Analytics and Cognition at La Trobe University, Australia. His research interests include hyperdimensional computing, neuromorphic computing, cognitive computing, graph neural networks, energy AI and computer vision.



Dilantha Haputhanthri completed a B.S. degree (Hons.) in Computer Science and a M.S. degree (Research) in Data Science in the Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka. He is currently pursuing a Ph.D. in Artificial Intelligence in the Centre for Data Analytics and Cognition at La Trobe University, Australia. His research interests include sparse representations, hyperdimensional computing, cognitive computing, and deep learning.



Thimal Kempitiya received the B.S. degree (Hons.) from the Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka. He is currently pursuing a Ph.D. in Artificial Intelligence in the Centre for Data Analytics and Cognition, at La Trobe University, Australia. Prior to commencing his Ph.D. degree, he worked as a Technical Lead in a Virtual Reality Healthtech start-up. His research interests include cognitive computing, autonomous systems, digital twins, and data stream mining.



Daswin De Silva (Senior Member, IEEE) received the Ph.D. degree in AI from Monash University, Melbourne, Australia. He is the Deputy Director and Course Architect of the Centre for Data Analytics and Cognition, La Trobe University, Australia. His research interests include Data Analytics, Artificial Intelligence (AI), Automation and the Ethics of AI, Data and Machines; with specific focus on autonomous learning, cumulative learning, incremental learning, active perception, information fusion, cognitive computing, NLP, deep emotions, psycholinguistics, and applications in industrial informatics, health, transport, energy, and smart cities. He is an Associate Editor of the following journals, IEEE Transactions on Industrial Informatics, PLOS One, IEEE Open Journal of the Industrial Electronics Society and Springer Discover AI. He was the Lead General Chair of the 15th IEEE International Conference on Human System Interaction - HSI 2022, Melbourne, Australia.



Daminda Alahakoon (Member, IEEE) received the Ph.D. degree in AI from Monash University, Melbourne, Australia. He is currently a Full Professor and the Founding Director of the Centre for Data Analytics and Cognition, La Trobe University, Melbourne, Australia. He has made significant contributions with international impact toward the advancement of AI through academic research, applied research, research supervision, industry engagement, curriculum development, and teaching. He has published over 100 research articles; theoretical research in self-structuring AI, human-centric AI, cognitive computing, deep learning, optimization, applied AI research in industrial informatics, smart cities, robotics, intelligent transport, digital health, energy, sport science, and education.



Denis Kleyko (Member, IEEE) received the Ph.D. degree in computer science from the Luleå University of Technology, Luleå, Sweden, in 2018. He is currently a Post-Doctoral Researcher on a joint appointment between the Redwood Center for Theoretical Neuroscience at the University of California at Berkeley, CA, USA and the Intelligent Systems Lab, Research Institutes of Sweden, Kista, Sweden. His current research interests include machine learning, reservoir computing, and vector symbolic architectures/hyperdimensional computing.