KI - Künstliche Intelligenz
https://doi.org/10.1007/s13218-019-00623-z

TECHNICAL CONTRIBUTION

An Introduction to Hyperdimensional Computing for Robotics

Peer Neubert[1] · **Stefan Schubert**[1] · **Peter Protzel**[1]

Received: 15 December 2018 / Accepted: 11 September 2019

Abstract
Hyperdimensional computing combines very high-dimensional vector spaces (e.g. 10,000 dimensional) with a set of carefully designed operators to perform symbolic computations with large numerical vectors. The goal is to exploit their representational power and noise robustness for a broad range of computational tasks. Although there are surprising and impressive results in the literature, the application to practical problems in the area of robotics is so far very limited. In this work, we aim at providing an easy to access introduction to the underlying mathematical concepts and describe the existing computational implementations in form of vector symbolic architectures (VSAs). This is accompanied by references to existing applications of VSAs in the literature. To bridge the gap to practical applications, we describe and experimentally demonstrate the application of VSAs to three different robotic tasks: viewpoint invariant object recognition, place recognition and learning of simple reactive behaviors. The paper closes with a discussion of current limitations and open questions.

Keywords Hyperdimensional computing · Vector symbolic architectures · Robotics

1 Introduction

Humans typically gain an intuitive understanding of 2-D and 3-D Euclidean spaces very early in their lives. Higher dimensional spaces have some counterintuitive properties that render the generalization of many algorithms from low to high-dimensional spaces useless—a phenomenon known as curse of dimensionality. However, there is a whole class of approaches that aims at *exploiting* these properties. These approaches work in vector spaces with thousands of dimensions and are referred to as hyperdimensional computing or vector symbolic architectures (VSAs) (previously they were also called high-dimensional computing or hypervector computing). They build upon a set of carefully designed operators to perform symbolic computations with large numerical vectors.

Another, better known class of algorithms that (internally) work with high-dimensional representations are (deep) artificial neural networks (ANN). Their recent success includes robotic subproblems, e.g., for robust perception. However, in many robotic tasks, deep learning approaches face (at least) three challenges [35]: (1) limited amount of training data, (2) often, there is prior knowledge that we want to integrate (models as well as algorithms), and (3) we want to be able to assess the generalization capabilities (e.g. from one environment to another or from simulation to real world). The later is particularly important if the robot is an autonomous car. A resulting motivation for using VSAs is to combine the versatility, representational power and noise robustness of high-dimensional representations (for example learned by ANNs) with sample-efficient, programmable and better interpretable symbolic processing.

Although processing of vectors with thousands of dimensions is currently not very time efficient on standard CPUs, typically, VSA operations can be highly parallelized. Further, VSAs support distributed representations, which are exceptionally robust towards noise [2], an omnipresent problem in robotics [36]. In the long term, this robustness can also allow to use very power efficient stochastic devices that are prone to bit errors but extend the battery life of a mobile robot [31].

The goal of this paper is to provide an easily accessible introduction to this field that spans the range from the mathematical properties of high-dimensional spaces in Sect. 2,

✉ Peer Neubert
Peer.Neubert@etit.tu-chemnitz.de

Stefan Schubert
Stefan.Schubert@etit.tu-chemnitz.de

Peter Protzel
Peter.Protzel@etit.tu-chemnitz.de

[1] Chemnitz University of Technology, Chemnitz, Germany

Published online: 18 September 2019

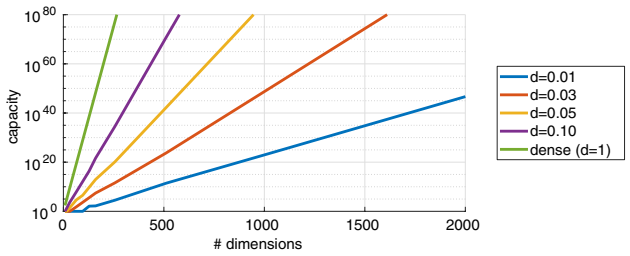

Fig. 1 Capacity of dense and sparse vector spaces quickly becomes very large (d is the ratio of ones). A discussion of properties of sparse representations can, e.g., be found in [2]

over implementations and computing principles of VSAs in Sect. 3, and a short overview of existing applications in Sect. 4, to three (novel) demonstrations how hyperdimensional computing can address robotic problems in Sect. 5. These demonstrations are intended as showcases to inspire future applications in the field of robotics. Remaining impediments in form of current limitations and open questions are discussed in Sect. 6.

2 Properties of High-Dimensional Spaces: Curse and Blessing

2.1 High-Dimensional Spaces Have Huge Capacity

The most obvious property is high capacity. For example, when we increase the number of dimensions in a binary vector, the number of stored possible patterns increases exponentially. For n dimensions, the capacity is 2^n. For real valued vector spaces and practical implementations with limited accuracy (i.e. a finite length representation in a computer) the capacity is also exponential in the number of dimensions. Interestingly, even for *sparse* binary vector spaces, the number of possibly stored patterns grows very fast. Figure 1 illustrates this behavior. For n dimensions and density d (the rate of ones in the vector), the capacity is $\binom{n}{\lfloor d \cdot n \rceil}$. Even if there are only 5% non-zero entries, a 1000 dimensional vector can store more patterns than the supposed number of atoms in the universe (presumably about 10^{80}).

2.2 Nearest Neighbor Becomes Unstable or Meaningless

This is a less intuitive property but nevertheless it is very important since it lies at the heart of the *curse of dimensionality*. This term was coined by Bellman [3] to describe the downsides of the exponential growth of capacity (or volume) of the space: if there is a fixed number of known data points (e.g. training samples), the sampling density decreases with increasing number of dimensions. For an n-dimensional space and k samples, it is proportional to $k^{1/n}$ (cf. [11, p. 23]). If we require 100 samples for an accurate representation of a one dimensional problem, the same problem in a 10 dimensional space would require 100^{10} samples to achieve the same sample density.

Beyer et al. [4] showed a direct consequence for the nearest neighbor problem (given a set of data points in an n-dimensional metric space, the task is to find the closest data point to some query point). They define a query as unstable if the distance from the query point to most datapoints is less than $(1 + \epsilon)$ times the distance from the query to the nearest neighbor. Under a broad range of practically relevant conditions, for any fixed $\epsilon > 0$ and increasing number of dimensions, the probability that a query is unstable converges to 1. In other words, the distance to the nearest neighbor approaches the distance to the farthest data point.

Based on the these results on the contrast in nearest neighbor queries in high-dimensional spaces, Aggarwal et al. [1] investigated the influence of the choice of the metric. For example, the often used Euclidean L_2 norm is not well suited for high-dimensional spaces, better choices are L_p norms with smaller p (for some applications this includes fractal norms with $p < 1$). Also angular distances for real vectors and Hamming distance for binary vectors are suitable choices.

2.3 Random Vectors are Very Likely Almost Orthogonal

Random vectors are created by sampling each dimension independently and uniformly from the underlying space. The distribution of angles between two such random vectors contradicts our intuition. In an n-dimensional real valued space, for any given vector, there are $n - 1$ exactly orthogonal vectors. However, the number of *almost orthogonal* vectors, whose angular distance to the given random vector is $\leq \frac{\pi}{2} + \epsilon$, grows exponentially for any fixed $\epsilon > 0$. An

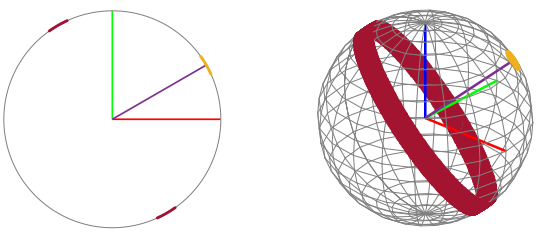

Fig. 2 Example visualization for similar and almost orthogonal areas in 2-D and 3-D spaces (angular thresholds 0.1)

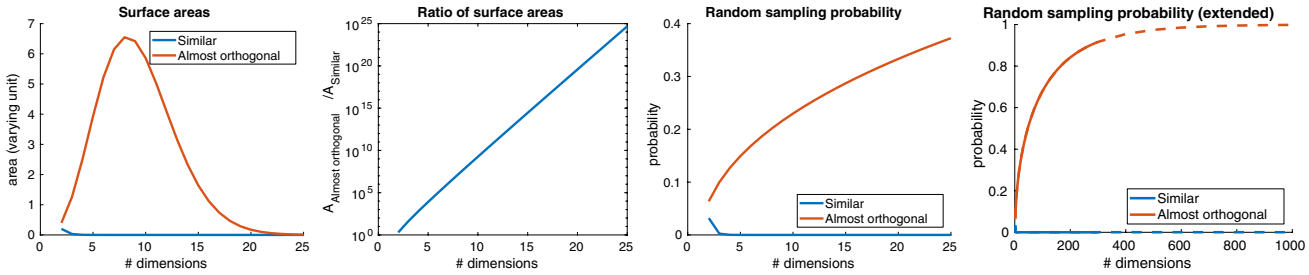

Fig. 3 Analytical results on n-spheres. Note the logarithmic scale in the second plot. Due to numerical reasons, the dashed extension for $\#dimensions > 300$ in the right plot is not obtained analytically but using sampling

important consequence is that two randomly chosen vectors are very likely to be almost orthogonal.

Although we can not directly illustrate this effect in high-dimensional spaces, the transition from 2-D to 3-D space already gives an idea of what is happening. Figure 2 shows the sets of similar and almost orthogonal vectors on unit spheres in 2-D and 3-D space. It can be easily seen that in the higher dimensional space, a random point on the sphere is much more likely to lie in the red area of almost orthogonal vectors than in the yellow area of similar vectors, and that this effect increased from 2-D to 3-D.

Figure 3 shows analytical results for higher dimensional spaces based on the equations of surface areas of n-spheres and n-caps.[1] The first plot shows the surface areas of the similar and almost orthogonal ranges for an angular distance for $\epsilon = 0.1$ (the similar and almost orthogonal one and two dimensional surfaces from Fig. 2 provide each two points on the corresponding curves in this first plot).[2] Although the value of the surface area also decreases beyond the local maximum (which is also a global maximum), it decreases much slower than the area value of the similar region. This is demonstrated in the second plot in Fig. 3 that shows the ratio of the almost orthogonal and similar surface areas. The linear shape in this logarithmic plot reveals the exponential growth of this ratio. The two right plots show the probability to randomly sample either a similar or an almost orthogonal vector. For high number of dimensions (e.g. > 700) the probability that two random vectors are almost orthogonal ($\epsilon = 0.1$) gets close to 1.

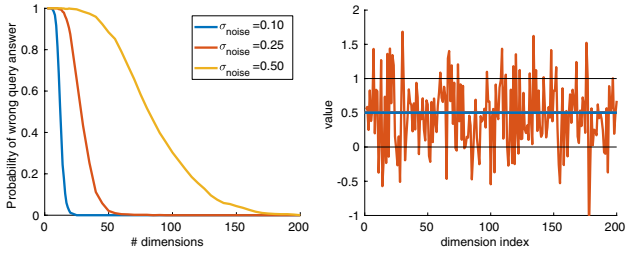

Fig. 4 (Left) Robustness towards different noise. (Right) For illustration: the blue example database vector $[\frac{1}{2}, \frac{1}{2}, \ldots]$ is affected by additive noise $\sim \mathcal{N}(0, 0.5)$ (the amount represented by the yellow curve on the left) and becomes the red vector (color figure online)

2.4 Noise has Low Influence on Nearest Neighbor Queries with Random Vectors

Why is it important that random vectors are very likely almost orthogonal? If random vectors point in almost orthogonal directions, this creates a remarkably robustness when trying to recognize them from noisy measurements. Let us demonstrate this with an example:[3] suppose there is a database of one million random feature vectors $f_i \in \mathbb{V} = [0, 1]^n$ (again, each dimension is sampled independently from a uniform distribution). Also there is a query vector, which is a noisy measurement of one of the database vectors (each dimension has additive noise $\sim \mathcal{N}(0, \sigma)$, as a consequence they can also leave their original range [0, 1]). Using the angular distance, what is the probability to get a wrong query answer (i.e. that a wrong vector from the database is closer to the noisy query than the correct one)? Figure 4 shows results for increasing number of dimensions and increasing amounts of noise. Even for noise with standard deviation of 0.5, that is half the available range of the initial value (this is illustrated in the right part of Fig. 4), using

[1] n-sphere: a hypersphere in the (n+1)-dimensional space. n-cap: portion of an n-sphere cut off by a hyperplane.

[2] Please keep in mind, that the unit of the surface area of an n-sphere is an n-dimensional object, thus the unit along the vertical axis changes and the values along the curves are not directly comparable. Nevertheless, the fact that there is a local maximum of the surface area of the almost orthogonal range is surprising. However, it is a direct consequence of the local maximum of the surface area of the whole unit n-sphere (which in turn becomes intuitive based on the recursive expression of the surface area $A_{n+1} = A_n \cdot \frac{n}{2\pi}$ since for $n > 2\pi$ this factor becomes smaller one).

[3] Similar experiments can, e.g., be found in [31] and [2]; analytical results on VSA capacity can be found in [7].

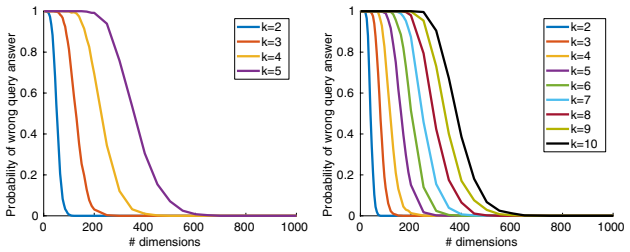

Fig. 5 Performance using bundles. Correct query answers are those were *all* k-nearest neighbors are correct. (left) Random vectors are from $\mathbb{V} = [0, 1]^n$ (right) $\mathbb{V} = [-1, 1]^n$

more than about 170 dimensions renders the probability of a false matching almost zero.

Adding noise to each dimension is the same as adding a noise vector to the the whole data vector. This yields an interesting application: What if this added vector is again a known data vector? This is known as the *bundle* of two vectors and will be subject of Sect. 3 where we will use the symbol + to refer to this operator. Since both vectors act symmetrically as noise for the recognition of the other, a query with the sum of two data vectors is expected to return both vectors as the two nearest neighbors. Figure 5 shows results for this experiment using the above database of one million data vectors and also different numbers k of summed vectors. We evaluate the capability to return perfect query answers (i.e. exactly the k true vectors are the k-nearest neighbors). As a reading example for these plots: The purple curve in the left plot shows that in a 600 dimensional vector space, we can safely add five vectors and the result is almost certainly more similar to all of these five vectors than to any other of the one million data vectors from our example database.

This is the most straightforward way of implementing bundling and we can easily improve performance. E.g., to allow for increasing and decreasing values during summation, we could sample each dimension from $[-1, 1]$ instead of $[0, 1]$ as before. The result are shown in the right part of Fig. 5. In this configuration, 300 dimensions are sufficient to recognize each of a sum of five vectors, and a 600 dimensional sum can handle a sum of ten vectors. Presumably, there are many other ways to improve performance in this simple example.

The next section will present a more systematic approach to solve problems with hyperdimensional computing that build upon the presented properties of high-dimensional spaces. While the examples from this section build upon random vectors, Sect. 5 will demonstrate performance of these approaches when confronted with data from robot sensors.

3 How to do Hyperdimensional Computing: Vector Symbolic Architectures (VSA)

The previous Sect. 2 listed properties of high-dimensional vector spaces and demonstrated how a bundle of vectors can be represented by their sum. Formally, the resulting vector represents the unordered set of the bundled vectors. To be of broader value, we need to be able to represent more complex and compositional structures such as ordered lists, hierarchies, or object-part relations. A key element to storing structured data is to assign different roles to different parts of the data [15]. Think of the personal record: {name = Alice, year_of_birth = 1980, high_score = 1000}. Storing just the values { Alice, 1980, 1000} is of limited help, since, for example, this unordered set cannot distinguish between Alice's year of birth and her high score. We need information about the binding between the role (or variable) "year_of_birth" and its filler (or value) "1980".

There have been multiple approaches presented to do this using hyperdimensional computing, e.g. by Plate [28], Kanerva [15], Gayler [8], and others. In 2003, Gayler coined the term *Vector Symbolic Architectures* (VSA) for these approaches [9]. In a nutshell, a VSA combines a vector space with a set of carefully chosen (designed) operators with particular properties. Each of the above VSAs uses a different vector space. The set of operators has to include the two operators bundling + and binding $\otimes$, and for certain applications a permute (or protect) Π operator is required. The output of each operator is again a vector from the same space. Bundling shares some properties with addition and binding some properties with multiplication of numbers.

Before we proceed with the formal requirements, let us give an example of the overall goal. Given are vector representations $Alice_V$, 1980_V and 1000_V, of the string "Alice" and the two numbers "1980" and "1000"; they can be obtained using a suitable encoder or be just random vectors whose meaning we have stored for later decoding. Also, we have random vectors $name_V$, $year\_of\_birth_V$, and $high\_score_V$ that represent the corresponding roles. Using hyperdimensional computing, we want to be able to create a *single* vector H that contains the whole record using the above operators:

$$\begin{aligned} H = {} & name_V \otimes Alice_V \\ & + year\_of\_birth_V \otimes 1980_V \\ & + high\_score_V \otimes 1000_V \end{aligned} \tag{1}$$

Subsequently, we want to be able to query for the value of each part of the composite structure using the same operators, e.g. for the name:

$$H \otimes name_V \rightarrow Alice_V \tag{2}$$

Table 1 Example VSAs

	Smolensky [32]	Plate [28]	Kanerva [15]	Gayler [8]
Space $\mathbb{V}$	Tensors of real numbers	Real and complex vectors	$\{0,1\}^n$	$[-1,1]^n$ (or $\{-1,1\}^n$)
Bundle $+$	Elementwise sum	Elementwise sum	Thresholded elementwise sum	Limited elementwise sum
Bind $\otimes$	Tensor product[a]	Circular convolution	Elementwise XOR	Elementwise product
Protect Π	(Not considered)	(Not considered)	Permutations	Permutations

A more extensive list can be found in [31]

[a]This operator changes the vector size and shape

This example will be explained in the following subsections. Querying a record is a simple example of high-dimensional computing. Before we proceed with more sophisticated demonstrations in Sects. 4 and 5, we will characterize the operators in more detail and explain how the properties of high-dimensional spaces are exploited in the query-record example. However, there is no exact definition available of the required properties of the VSA operators, the exact set of operators or the vector space $\mathbb{V}$. The following is a summary of properties from the literature, i.e. the VSAs from Table 1.

3.1 Binding $\otimes$

Binding $\otimes : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$ combines two input vectors into a single output vector that is *not similar* to the input vectors but allows to (approximately) *recover* any of the input vectors given the output vector and the other input vector. E.g., we can bind the filler $Alice_V$ to the role $name_V$ by $N = Alice_V \otimes name_V$ and later recover the filler $Alice_V$ given N and the role vector $name_V$. To recover this vector, we need to unbind one vector from another. For VSAs were vectors are also (approximately) self inverse, unbinding and binding are the same operation (e.g. [8, 15]). Self-inverse means:

$$\forall X \in \mathbb{V}: \quad X \otimes X = \mathbf{1}$$

where $\mathbf{1}$ is the neutral element of binding in the space $\mathbb{V}$. We will use this property in the following examples.

An intuitive example of such a binding operator is the special case of Gayler's VSA with $\mathbb{V} = \{-1,1\}^n$ (instead of $[-1,1]^n$) and binding and unbinding as elementwise multiplication. The self-invertibility is due to the limitation to ± 1, since $-1 \cdot -1 = 1 \cdot 1 = 1$ and 1 is the neutral element of multiplication. With such a VSA, recovering the name in the above role-filler example works as follows:

$$\begin{aligned} N \otimes name_V &= (Alice_V \otimes name_V) \otimes name_V \\ &= Alice_V \otimes (name_V \otimes name_V) \\ &= Alice_V \otimes \mathbf{1} = Alice_V \end{aligned}$$

This example also requires the binding operator to be associative. Further, to allow to change the order of the vectors, binding is typically also commutative. Table 1 lists several available binding implementations. Section 2 illustrated that the distribution of similar and dissimilar vectors is an important property of high-dimensional vectors spaces. Thus, the effect of VSA operations on these similarities is also important. E.g., binding should be similarity preserving: $\forall A, B, X \in \mathbb{V} : dist(A,B) = dist(A \otimes X, B \otimes X)$, the distance of two vectors remains constant when binding both vectors to the same third vector. Moreover, as the first sentence of this section already said, the result vector has to be dissimilar to the two inputs. This is important for the combination with the bundle operator explained in the following section.

3.2 Bundling $+$

The goal of the bundling operation $+ : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$ is to combine two input vectors such that the output vector is similar to both inputs. This is also called superposition of vectors. Typically, the bundling operator is some kind of elementwise sum of the vector elements (see Table 1). E.g., the Multiply–Add–Permute VSA of Gayler [8] uses elementwise sum on the same vector space $[-1,1]^n$ as the experiments from Fig. 5 (the sum is limited to the range of the vector space elements $[-1,1]$). In these experiments, we already showed that the elementwise sum of vectors is similar to each of the vectors; this was a direct consequence of the almost orthogonality of random vectors.

According to Kanerva [17] the bundle and bind operations should "form an algebraic field or approximate a field". In particular, bundling should be associative and commutative and binding should distribute over bundling. Let us illustrate this with a closer look at the example of Alice's record. For brevity we use X, Y, Z for the role vectors "*name*", "*year_of_birth*" and "*high_score*" and A, B, C for the vector representations of their values "*Alice*", "1980" and "1000". The record vector is formed by $H = (X \otimes A) + (Y \otimes B) + (Z \otimes Z)$. What happens when querying for the name by binding with its vector X?

$$\begin{aligned} X \otimes H &= X \otimes ((X \otimes A) + (Y \otimes B) + (Z \otimes Z)) \\ &= (X \otimes X \otimes A) + (X \otimes Y \otimes B) + (X \otimes Z \otimes C) \\ &= A + noise \end{aligned}$$

The *noise* term includes the terms $(X \otimes Y \otimes B)$ and $(X \otimes Z \otimes C)$. Both are non-similar to each of their elements (a property of binding). Thus the only known vector that is similar to $X \otimes H$ is A. Again this exploits the property of high-dimensional vectors to be almost sure non-similar (i.e. almost orthogonal) to random vectors. The database experiments from Sect. 2.4 already illustrated how a noise-free version of vector A can be recovered: given a database with all elementary vectors, returning the nearest neighbor to $A + noise$ results very likely in A (to not return a vector which is similar to *noise*, we need the property of binding to be non-similar to its inputs). In VSAs this database is typically called *clean-up* or *item memory* [16]. It can be as simple as our look-up table or, e.g., an attractor network [17]. Section 5.2 will evaluate properties of such a clean-up memory in combination with real-world data.

There can be trade-offs between the performance of the bundling and binding operators. For example, in the VSA of Gayler, the bundling operator works well for $\mathbb{V} = [-1, 1]^n$; however, the self-inverse property of binding holds only exactly for the special case of $\mathbb{V} = \{-1, 1\}^n$. The clean-up memory can also be used to restore exact values in the non-exact inversion case.

3.3 Permutation (or Protect) Π

Gayler [8] discussed the benefit of using an additional operator in order to protect vectors. Think of a situation with two bound role-filler pairs: $A \otimes X$ and $B \otimes Y$. When binding these two pairs to $(A \otimes X) \otimes (B \otimes Y)$, it becomes necessary to prevent mixing roles and fillers: Since binding is associative and commutative, this is equivalent to $(A \otimes Y) \otimes (B \otimes X)$. The permutation operator Π protects a term from associative and distributive rules. In the above example this is $(A \otimes X) \otimes \Pi(B \otimes Y)$. It is typically implemented as a permutation of vector dimensions. Its output is dissimilar to the input and by application of the reverse permutation, it is also invertible. For details please refer to [8].

4 Applications from the Literature

VSAs have been applied to various problems like text classification [20], fault detection [19], analogy mapping [30], and reinforcement learning [18]. Kanerva [17] discusses the general computational power of VSAs and concludes one could create a "High dimensional computing-Lisp". While this is still open, work in this direction includes synthesis of finite state automata [27] and hyperdimensional stack machines [38]. Danihelka et al. [5] (Deepmind) used a VSA to model long-short term memory. In the medical domain, Widdows and Cohen [37] used Predication-based Semantic Indexing which exploits a VSA to represent traditional subject-predicate-object relationships (e.g. "aspirin TREATS headache") for fast approximate inference on the relationships of diseases, symptoms and treatments. Natural language processing is considered a challenging task. Jackendoff [13] specified this statement to four theoretical challenges that a system that aims at processing language at a human level has to solve. According to Gayler [9], VSAs can solve these challenges. Hyperdimensional computing was also used to encode n-gram statistics to recognize the language of a text [14]. There is evidence that distributed high-dimensional representations are widely used for representation in the human brain [2]. This is extensively used in brain-inspired cognitive systems like Spaun [6] and in hierarchical temporal memory (HTM) [12], a computational model of working principles of the human neocortex. The latter was also applied for mobile robot place recognition [24].

5 Application to Robotic Tasks

This section showcases three examples, how hyperdimensional computing can be used for real robotic tasks. We do not claim that the presented approaches are better than existing solutions to the considered tasks, however, they demonstrate the versatility of hyperdimensional computing, its capability to work with real world data and advocate the practical value. Before we start with the applications, we will describe how we bridge the gap between real world sensors and vector computations.

5.1 Encoding Real World Data

Section 2.4 used synthetic data to demonstrate the noise robustness of hyperdimensional computations and its application to bundling. The random vectors in this synthetic data fulfill the requirements to achieve pairwise almost orthogonal vectors by design. What if we want to work with real world data that does not provide thousands of independent random dimensions? For simple data structures and the particular case of sparse binary vectors, Purdy [29] discusses different encodings. Very recently, Kleyko et al. [20] discussed trade-offs in binary hyperdimensional encodings of images. A comprehensive discussion of encoding approaches of real world sensor data is beyond the scope of this paper. However, we want to shortly describe our approach to encode the real world image data in our experiments.

Any high-dimensional image feature vector can potentially be used. Based on their recent success, we decided to use image descriptors from early layers from deep convolutional neural networks in a similar fashion as they are used for place recognition [25, 33]. To get a descriptor for an image, it is fed to an off-the-shelf readily trained CNN

Fig. 6 Example views of one of the 1000 ALOI objects from 0°, 90° and 180° viewing angle

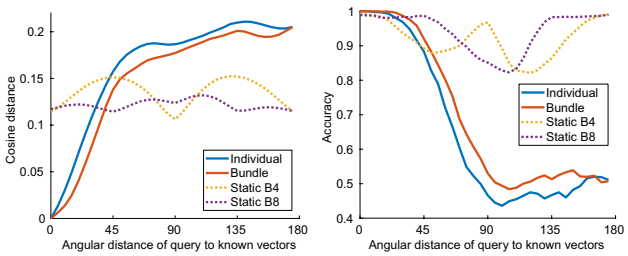

Fig. 7 Object recognition performance on ALOI dataset (color figure online)

(we use AlexNet [21]) and instead of using the final output (e.g. the 1000 dimensional soft-max class output), the intermediate output of an earlier layer is used [we use the $13 \times 13 \times 385 = 64{,}896$ dimensional output of the third convolutional layer (conv3)]. To reduce computational effort and to get a distributed representation, we use a locality sensitive hashing (LSH) approach and project the normalized conv3 descriptor with a random matrix R to a lower dimensional space. Each row in R is the normal of a 64,896 dimensional hyperplane (obtained by sampling R from a standard normal distribution followed by normalization of rows to length one). Since these products of the normalized conv3 descriptor and each row (hyperplane normal) reflect the cosine of the angle between the vectors, they are in range $[-1, 1]$ and can be directly used in the Multiply–Add–Permute Architecture [8] (see Table 1). We use 8192 rows in R.

5.2 Bundling Views for Object Recognition

Robotic task For the first robotic use-case, we demonstrate the application of hyperdimensional computing to recognize objects from multiple viewpoints. This is important for mobile robot localization by recognizing known landmarks, recognizing objects for manipulation, and other robotics tasks.

Motivation We use this task to transfer the results on synthetic data from Sect. 2.4 on bundling of high-dimensional vectors to real world data. Bundling allows to combine multiple vectors into one. This can be straightforwardly used to combine two or more known views. The motivation is threefold: (1) if we combine all known views into one representation, the comparison of a query vector to all known representations is a single vector comparison. (2) There might be a better interpolation between the known views. (3) This allows to straightforwardly update the representation of an object, particularly iteratively in an online-filter fashion.

Experimental setup We practically demonstrate this approach using the Amsterdam Library of Object Images (ALOI) dataset [10], in particular the collection of 72,000 images of 1000 objects seen from 72 different horizontal viewing angles (5° steps). Figure 6 shows example images. In our experiments, given are a database of $k \in \{1 \dots 1000\}$ known images I_x^k and I_y^k at viewing angles x and y, as well as a query image I_z^q at viewing angle $z = \frac{x+y}{2}$ (the viewing angle in between) and image index q. The task is to assoziate q to the correct image index k.

VSA approach We bundle the image descriptors I_x^k and I_y^k, i.e. create $I_x^k + I_y^k$ for each k (there will be one vector for each object in the database).

Results When comparing a query image I_z^q to the database, motivation (1) is achieved by design: instead of comparing I_z^q to I_x^k and I_y^k individually, we can now compare against the single bundle vector and reduce the number of required comparisons by factor two.

The results in Fig. 7 demonstrate the better interpolation capabilities from motivation (2): the bundled representation (red curve) has a smaller cosine distance to the object image under a novel viewing angle than the individual images (blue curve). This also results in a better object recognition accuracy (right part). See footnote[4] for details.

To evaluate towards continiously integrating more views [motivation (3)], the yellow and the purple curves in Fig. 7 show query results when bundling a (static) set of multiple views from the four angles $\{0, 90, 180, 270\}$ and the eight angles $\{0, 45, 90, \dots, 315\}$. Although the distance values

[4] Details: the red curve in the left plot evaluates vector similarities (the query image index q is known and we compare the similarity of $I_x^k + I_y^k$ and $I_z^{q=k}$), the red curve in the right plot evaluates the accuracy of a nearest neighbor query (the query image index q is not known to the system and it returns the index k of the nearest neighbor to I_z^q of all $I_x^k + I_y^k$, $k \in \{1 \dots 1000\}$). x is fixed at viewing angle 0°. y varies from 0° to 350°. The horizontal axis is the mean angular distance from z to x and y. As a reading example: in the left plot, the red curve evaluated at 90° means that for $x = 0°$, $y = 180°$, $z = 90°$ (e.g. the images from Fig. 6), the average cosine distance of the bundle $(I_0^k + I_{180}^k)$ and I_{90}^k is about 0.17, and the right plot tells us that for about 53% of the objects the query image was most similar to the correct bundle. For comparison without bundling, the blue curves in Fig. 7 show the results when comparing the query image to the individual images I_x^k and I_y^k (instead of their bundle). For the distance evaluation in the left plot, we use the *closest* of the two individual results for each query. For the query results in the right plot, all views I_x^k and I_y^k are stored in the database and a single query is made (the number of data base entries and thus comparisons has now doubled compared to the bundling approach). The VSA approach not only reduces the number of comparison, it also performs slightly better than using individual comparisons in both plots.

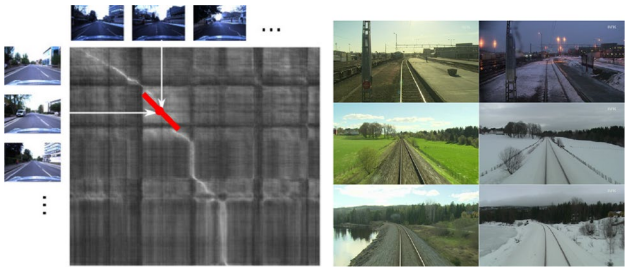

Fig. 8 (Left) Illustration of a similarity matrix and SeqSLAM post-processing. (Right) Three example places from the Nordland dataset in spring and winter

does not reach zero distance for known views for these larger bundles, the cosine distance varies the less the more views are bundled.

5.3 Sequence Processing for Place Recognition

Robotic task Place recognition is a task similar to image retrieval: given a set of images from known places, find corresponding places to the current camera view of the robot. In contrast to the more general image retrieval task, for place recognition, we can assume that the robot does not jump arbitrarily between places and that the sequence of previous places provide some information about the current place.

Motivation The previous application used only bundling. This second examples demonstrates the combination with the binding operator $\otimes$ to implement the important concept of role-filler pairs using hyperdimensional computing. We also use this example to showcase how hyperdimensional computing can provide a simple and concise implementation of an existing algorithm (SeqSLAM). This VSA implementation can also potentially benefit from the general advantages of hyperdimensional computing like noise tolerance and potential energy efficiency. Similar to the previous object recognition example, the superposition of vectors also reduces the number of required comparison operations.

Mimicked approach SeqSLAM [23] exploits image sequence information to approach the challenging problem of place recognition in changing environments (e.g. given a database of images taken in summer and the goal is to localize during winter). Input to Seq-SLAM's sequence processing part is a pairwise image similarity matrix S shown in Fig. 8. Each entry $s_{i,j}$ is the similarity between the ith image from the database and the jth image of the robot's current camera sequence. The output of SeqSLAM is a new value for the similarity of images i and j based on their similarity in S and the similarities between images *before and after* i and j. SeqSLAM assumes a constant velocity within each sequence, thus it sums over the similarities on a short *linear segment* in S centered at $s_{i,j}$ (illustrated as red line in Fig. 8). See [23] for more details.

VSA approach To implement the SeqSLAM idea using hyperdimensional computing, we first encode each image as a vector using the approach from Sect. 5.1. Then, the basic idea is to replace each image vector in each sequence by a vector (of the same dimensionality) that encodes this image and the d images before and after this image in a bundle. To preserve the order of the images, each image is bound to a static position vector P_k before bundling. In our experiments, the position vectors are random vectors, thus they are very likely almost orthogonal. The encoding of each image is: $Y_i = +_{k=-d}^{d}(X_{i+k} \otimes P_k)$; for the beginning and end of the sequence (e.g. $i < d + 1$), fewer vectors are bundled. Since the bundle of an arbitrary number of vectors has exactly the same shape, this is neatly handled. Finally, to obtain place recognition results, the Y_i encodings of the database and query image sets can be compared pairwise.

Why does this work? Consider the encodings of a sequence of two consecutive images from the database: $(A_a \otimes P_0) + (A_{a-1} \otimes P_{-1})$, and a sequence of two consecutive query images: $(B_b \otimes P_0) + (B_{b-1} \otimes P_{-1})$. When comparing these two bundles, they are the more similar, the more of their components are similar. Let us evaluate some component pairs: The similarity of $(A_a \otimes P_0)$ and $(B_b \otimes P_0)$ depends on the similarity of A_a and B_b since both are bound to the *same* vector P_0 and operator $\otimes$ is similarity preserving. The same holds for A_{a-1} and B_{b-1}. In contrast, e.g., $(A_a \otimes P_0)$ and $(B_{b-1} \otimes P_{-1})$ are known to be non similar since P_0 and P_{-1} are almost orthogonal.

Experimental setup We use the Nordland dataset [34] which provides images from four 728 km train journeys through Norway, once each season (see Fig. 8 for example images). We use 288 equally spaced places along the tracks and perform place recognition between the spring and the winter image sets. The evaluation is done using precision-recall curves based on the known ground-truth place associations.

Results The experimental results in Fig. 9 show that the VSA SeqSLAM implementation (solid curves) can exploit sequential information to improve the place recognition performance. The results closely approximate the results of the original SeqSLAM sequence processing approach (dashed curves).

There is an additional theoretical benefit: the number of performed operations is significantly smaller for the VSA approach. For database size n, a query size m, and sequence length $d_s = 2 \cdot d + 1$ (past + future + current), the number

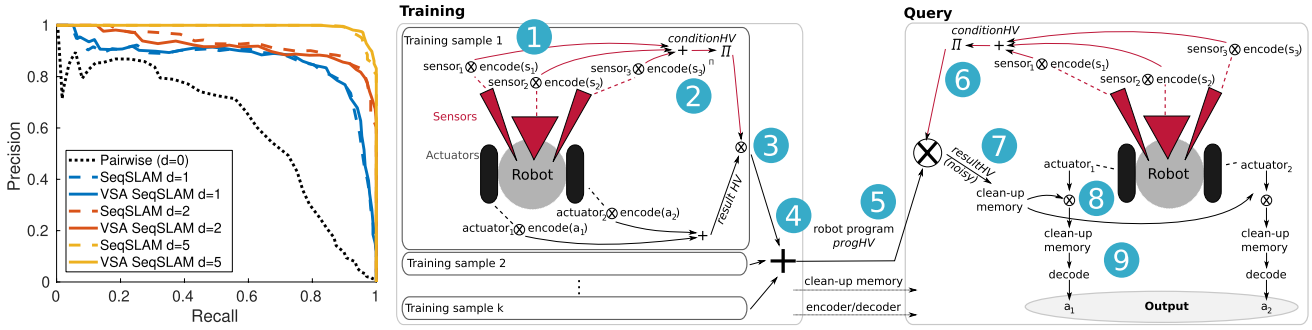

Fig. 9 (Left) Place recognition results on Nordland dataset. The original SeqSLAM sequence processing approach is well approximated by the vector sequence encoding. Both improve the place recognition performance compared to a direct pairwise comparison (top-right is better). (Right) Schematic overview of data flow for behavior learning

of original SeqSLAM operations is $m \cdot n \cdot d_s$. For the VSA implementation it is $m \cdot d_s + n \cdot d_s + m \cdot n$ (the first two terms represent the descriptor bundling and the last term the final pairwise comparison). E.g., for our database and query size of 288 images and $d = 5$, the ratio of the numbers of operations is more than factor 10 and it becomes larger if any of these values increases. Unfortunately, while for the original SeqSLAM most of the operations deal with scalar similarity values, for the VSA approach all operations are high-dimensional vector operations. Presumably, a *practical* runtime improvement can only be achieved with special hardware for high-dimensional vector computations (which then could also exploit the energy saving potential of VSAs).

Extensions The Nordland data is perfectly suited for SeqSLAM and its vector variant since each train journey is a single long sequence with constant speed. To account for a slightly varying speed between the sequences, the original SeqSLAM algorithm evaluates line segments with different slopes and uses the best choice. The proposed VSA implementation can be straightforwardly extended to these varying velocities by superposing the different combinations of image vectors and sequence position vectors. Further interesting directions would be to control the similarity between neighbored sequence position vectors or to use other VSAs' ways of encoding sequence information, e.g. permutations [16].

5.4 Learning and Recall of Reactive Behavior

Robotic Task The task is to learn simple reactive behaviors from demonstration. "Simple" means that we can represent them as a set of sensor-action (condition-result) pairs. Given a successful demonstration of a navigation run (e.g. from a human) by pairs of sensor input and actuator output, the system learns a representation that encodes this reactive behavior and can resemble it during new runs in the environment.

Motivation The goal of this final example is to showcase a more complex VSA-based system.[5] In contrast to the previous applications this does also involve action selection by the robot. The goal is to encode the whole robot program (a set of reactive behavior rules) in a single vector. When executing (and combining) such VSA-based behaviors, the advantages of vectors (i.e. the representational power and robustness to noise) are preserved. A particular beauty of this approach is that it can learn encodings for behaviors that have exactly the same form (a single vector) no matter how complex the sensor input or the behaviors are.

Experimental setup We use the simulation task described in [22]. Figure 9 illustrates the used simple robot with differential drive (i.e. a left and a right motor), a left and a right distance sensor, and a central light sensor. The robot starts at a random location in a labyrinth and the task is to wander around while avoiding obstacles, until the robot finds a light source. Then the robot should stay under this light. This is a simple task that can be coded using a simple set of rules (e.g. see [22]).

VSA approach The listing in Algorithm 1 describes the learning procedure. Inputs are pairs of sensor measures and corresponding actuator commands. The idea is to (1) encode the sensor and actuator values individually, (2) combine a sensor value in a condition vector and all actuator encodings in a result vector, (3) combine the condition with the result vector to a rule vector, and finally (4) combine all rule vectors to a single vector that contains the whole "program". Algorithm 2 is used in the execution phase to find the best actuator commands for the current sensor input.

[5] This work was previously presented at an IROS workshop, see [26] for details.

Algorithm 1: Learning

Data: k training samples $[S, A]_{1:k}$ of sensor and actuator values, a VSA, an encoder, an empty program progHV and an empty vector knownCondHV of known conditions
Result: progHV - a vector representation of the behaviour

// get vector representations for each sensor and actor
$[sensor, actuator] = VSA.assignRandomVectors()$
// for each training sample $[S, A]$
foreach *pair* $[S = (s_1, ..., s_n), A = (a_1, ..., a_m)]$ **do**
  // encode values, bind to device and bundle condition/result
  $conditionHV := +_{i=1}^{n}(sensor_i \otimes encode(s_i))$
  $resultHV := +_{i=1}^{m}(actuator_i \otimes encode(a_i))$
  if $isDissimilar(knownCondHV, \Pi(conditionHV))$ **then**
    // protect the condition and append (bundle) to the program
    $progHV := progHV + (\Pi(conditionHV) \otimes resultHV)$
    // also append (bundle) the condition to the set of known conditions
    $knownCondHV := knownCondHV + (\Pi(conditionHV))$
    // insert the result and the actuator encoding to the clean-up memory
    $VSA.addToCUM(resultHV)$
    foreach $actuator_i$ **do**
      $VSA.addToCUM(actuator_i \otimes encode(a_i))$
    end
  end
end

Algorithm 2: Query

Data: $progHV$ - the output of the learning procedure Alg. 1, the VSA and encoder/decoder used in Alg. 1, the query sensor inputs S
Result: output actuator commands A

// encode values, bind to device and bundle condition
$conditionHV := +_{i=1}^{n}(sensor_i \otimes encode(s_i))$
// query program to get a noisy version of the resultHV
$resultHVNoisy := \Pi(conditionHV) \otimes progHV$
// remove noise
$resultHV := vsa.queryCUM(resultHVNoisy)$
// for each actuator , extract the command from the result vector
foreach $actuator_i$ **do**
  // unbind a noisy version from the result vector
  $commandHVNoisy := actuator_i \otimes resultHV$
  // remove noise
  $commandHV := vsa.queryCUM(commandHVNoisy)$
  // decode the command value from the vector
  $a_i := decode(commandHV)$
end

Figure 9 illustrates how VSA operators are used during training and query. Each encoded sensor value is bound to a random vector that represents the corresponding sensor (see ① in Fig. 9). E.g., the left distance sensor has a random, but static vector representation (coined $sensor_1$) that indicates the role "left distance sensor". During training, for each input pair, all sensor-value-binding are bundled ②. The same happens on the actuator side. The complete sensor-action rule is stored as the binding of sensors and actuators ③. Since many of these rules are bundled to create the complete program ④, the condition vector has to be protected (think of it as using brackets that also prevent distribution in an equation) to prevent mixing up sensor conditions from different training pairs. To allow later recall from noisy vectors, each actuator-value pair [e.g. $actuator_1 \otimes encode(a_1)$] and the result bundle have to be stored in the clean-up memory.

In this example, during query, the task is to obtain the left and right motor commands given the current sensor input and the program vector. To be able to get the correct commands, also the encoder/decoder and the clean up memory are required ⑤. The given sensor information are combined to a condition vector as before ⑥. Binding this vector to the program vector retrieves the most similar rule vector from training ⑦. The clean-up memory can be used to obtain a noise-free version. By binding this result with an actuator role vector (e.g. $actuator_1$), a noisy version of the corresponding command is obtained ⑧. Using the clean-up memory and the decoder, this can be translated in a motor command and used to control the robot ⑨.

Results We implemented this system and were able to successfully learn behaviors that solve the described simulation task from [22] using human demonstration runs. For more details, please refer to [26].

This demonstrates that VSAs can also be used to implement more complex programs, including action selection. It is possible to encode a complete robot program in a single vector. However, the complexity of the program is limited by the capacity of this vector. More work is required to investigate the practical potential of this example.

6 Limitations, Discussion and Open Questions

We demonstrated that hyperdimensional computing and its implementation in form of VSAs have interesting properties and a variety of applications in the literature, and that they can be used to address robotic problems. However, there are a couple of shortcomings and potential limitations we want to discuss.

In our opinion, the first challenge is the lack of a clear definition of VSAs. It is a name for a collection of approaches to exploit the properties of high-dimensional vector spaces based on a set of operators with similar properties. We tried to collect information about different VSAs in Sect. 3 to provide a coherent definition. However, the list of properties of the operators includes terms like "should" or "approximately" without further ascertainment. The ultimate goal would be a theoretically rigorous definition in form of axioms and derived theorems about the capabilities of such systems.

There are trade-offs like the one between the binding and bundling operators in the Multiply–Add–Permute architecture explained in Sect. 3, the first works better using $\{0, 1\}^n$ as vector space, the other when using the interval $[0, 1]^n$. This is not an unscalable problem, neither in theory (e.g by using a clean-up memory) nor in practice (we also used this VSA in the robotics experiments in Sect. 5). Although some theoretical insights on properties of VSAs are available (e.g. on the bundle capacity [7]), better insights in such trade-offs and limitations would support the practical application.

A particularly important and challenging task is the encoding of real world data into vectors. Our examples in Sect. 2 and most applications from Sect. 4 use random vectors—which are very likely pairwise almost orthogonal. However, for the shown robotic experiments in Sects. 5.2 and 5.3, we used encodings obtained from images using a CNN and LSH (Sect. 5.1). The resulting vectors span only a subspace of the vectorspace. Thus, presumably, the VSA mechanisms work only approximately—nevertheless, they provide reasonable results. Insights to requirements on properties of the encoding/decoding could have a huge influence for practical application.

The fact that in hyperdimensional computing most things work only approximately, requires a different engineer's mind set. In the foreseeable future, complex machines like robots will very likely contain a lot of engineering work—an easier access for *non-mathematicians* to what works why and when in these systems would presumably be a very helpful contribution. A very interesting direction would also be the connection to the probabilistic methods that are widely used in this field.

Beside access to theoretical findings for applications of hyperdimensional computing, a structured way to practically designing systems using VSAs is missing. Currently, almost every problem that is solved using hyperdimensional computing is a somehow isolated application. Although the same principles are used on every occasion, a structured approach how to solve problems, e.g. by means of *design patterns*, would be very desirable. Also related is the very fundamental question, which parts of the system have to be designed manually and which parts can be learned. Currently, many results are due to elaborate design rather than learning. However, the high-dimensional representations presumably provide easy access to connectionists' learning approaches—potentially an elegant bridge between (deep) artificial neural networks and (vector) symbolic processing.

References

1. Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional space. In: Van den Bussche J, Vianu V (eds) Database theory—ICDT 2001. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 420–434
2. Ahmad S, Hawkins J (2015) Properties of sparse distributed representations and their application to hierarchical temporal memory. CoRR arxiv:abs/1503.07469
3. Bellman RE (1961) Adaptive Control Processes: A Guided Tour. MIT Press, Cambridge
4. Beyer K, Goldstein J, Ramakrishnan R, Shaft U (1999) When Is nearest neighbor meaningful? In: Beeri C, Buneman P (eds) Database theory—ICDT'99. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 217–235
5. Danihelka I, Wayne G, Uria B, Kalchbrenner N, Graves A (2016) Associative long short-term memory. In: Balcan MF, Weinberger KQ (eds) Proceedings of the 33rd international conference on machine learning, proceedings of machine learning research, vol 48. PMLR, New York, pp 1986–1994. http://proceedings.mlr.press/v48/danihelka16.html
6. Eliasmith C, Stewart TC, Choo X, Bekolay T, DeWolf T, Tang Y, Rasmussen D (2012) A large-scale model of the functioning brain. Science 338(6111):1202–1205. https://doi.org/10.1126/science.1225266. http://science.sciencemag.org/content/338/6111/1202
7. Frady EP, Kleyko D, Sommer FT (2018) A theory of sequence indexing and working memory in recurrent neural networks. Neural Comput 30(6):1449–1513. https://doi.org/10.1162/neco_a_01084
8. Gayler RW (1998) Multiplicative binding, representation operators, and analogy. In: Advances in analogy research: integr of theory and data from the cogn, comp, and neural sciences. Bulgaria
9. Gayler RW (2003) Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In: Proc. of ICCS/ASCS Int. Conf. on cognitive science, pp 133–138. Sydney, Australia
10. Geusebroek JM, Burghouts GJ, Smeulders AWM (2005) The Amsterdam library of object images. Int J Comput Vis 61(1):103–112
11. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference and prediction, 2 edn. Springer. http://www-stat.stanford.edu/~tibs/ElemStatLearn/
12. Hawkins J, Ahmad S (2016) Why neurons have thousands of synapses, a theory of sequence memory in neocortex. Front Neural Circ 10:23. https://doi.org/10.3389/fncir.2016.00023
13. Jackendoff R (2002) Foundations of language (brain, meaning, grammar, evolution). Oxford University Press, Oxford
14. Joshi A, Halseth JT, Kanerva P (2017) Language geometry using random indexing. In: de Barros JA, Coecke B, Pothos E (eds) Quantum interaction. Springer International Publishing, Cham, pp 265–274
15. Kanerva P (1997) Fully distributed representation. In: Proc. of real world computing symposium, pp 358–365. Tokyo, Japan
16. Kanerva P (2009) Hyperdimensional computing: an introduction to computing in distributed representation with high-dimensional random vectors. Cognit Comput 1(2):139–159
17. Kanerva P (2014) Computing with 10,000-bit words. In: 2014 52nd annual Allerton conference on communication, control, and computing (Allerton), pp 304–310 . https://doi.org/10.1109/ALLERTON.2014.7028470
18. Kleyko D, Osipov E, Gayler RW, Khan AI, Dyer AG (2015) Imitation of honey bees' concept learning processes using Vector Symbolic Architectures. Biol Inspired Cognit Arch 14:57–72. https://doi.org/10.1016/j.bica.2015.09.002
19. Kleyko D, Osipov E, Papakonstantinou N, Vyatkin V, Mousavi A (2015) Fault detection in the hyperspace: towards intelligent automation systems. In: 2015 IEEE 13th international conference on industrial informatics (INDIN), pp 1219–1224. https://doi.org/10.1109/INDIN.2015.7281909

20. Kleyko D, Rahimi A, Rachkovskij DA, Osipov E, Rabaey JM (2018) Classification and recall with binary hyperdimensional computing: tradeoffs in choice of density and mapping characteristics. IEEE Trans Neural Netw Learn Syst 29(12):5880–5898. https://doi.org/10.1109/TNNLS.2018.2814400
21. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Pereira F, Burges C, Bottou L, Weinberger K (eds) Advances in neural information processing systems, vol 25. Curran Associates, Inc., pp 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
22. Levy SD, Bajracharya S, Gayler RW (2013) Learning behavior hierarchies via high-dimensional sensor projection. In: Proc. of AAAI conference on learning rich representations from low-level sensors, AAAIWS'13–12, pp 25–27
23. Milford M, Wyeth GF (2012) SeqSLAM: visual route-based navigation for sunny summer days and stormy winter nights. In: Proceedings of the IEEE international conference on robotics and automation (ICRA)
24. Neubert P, Ahmad S, Protzel P (2018) A sequence-based neuronal model for mobile robot localization. In: Proc of KI: advances in artificial intelligence
25. Neubert P, Protzel P (2015) Neubert P, Protzel P (2015) Local region detector+ CNN based landmarks for practical place recognition in changing environments. In: Proceedings of the European conference on mobile robotics (ECMR)
26. Neubert P, Schubert S, Protzel P (2016) Learning vector symbolic architectures for reactive robot behaviours. In: Proc of Intl Conf on intelligent robots and systems (IROS) workshop on machine learning methods for high-level cognitive capabilities in robotics
27. Osipov E, Kleyko D, Legalov A (2017) Associative synthesis of finite state automata model of a controlled object with hyperdimensional computing. In: IECON 2017—43rd annual conference of the IEEE industrial electronics society, pp 3276–3281. https://doi.org/10.1109/IECON.2017.8216554
28. Plate TA (1994) Distributed representations and nested compositional structure. Ph.D. thesis, Toronto, Ont., Canada, Canada
29. Purdy S (2016) Encoding data for HTM systems. CoRR arxiv:abs/1602.05925
30. Rachkovskij DA, Slipchenko SV (2012) Similarity-based retrieval with structure-sensitive sparse binary distributed representations. Comput Intell 28(1):106–129. https://doi.org/10.1111/j.1467-8640.2011.00423.x
31. Rahimi A, Datta S, Kleyko D, Frady EP, Olshausen B, Kanerva P, Rabaey JM (2017) High-dimensional computing as a nanoscalable paradigm. IEEE Trans Circ Syst I Regular Pap 64(9):2508–2521. https://doi.org/10.1109/TCSI.2017.2705051
32. Smolensky P (1990) Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artif Intell 46(1–2):159–216
33. Sünderhauf N, Dayoub F, Shirazi S, Upcroft B, Milford M (2015) On the performance of ConvNet features for place recognition. CoRR arxiv:abs/1501.04158
34. Sünderhauf N, Neubert P, Protzel P (2013) Are we there yet? Challenging SeqSLAM on a 3000 km journey across all four seasons. In: Proceedings of the IEEE international conference on robotics and automation (ICRA), workshop on long-term autonomy
35. Sünderhauf N, Brock O, Scheirer W, Hadsell R, Fox D, Leitner J, Upcroft B, Abbeel P, Burgard W, Milford M, Corke P (2018) The limits and potentials of deep learning for robotics. Int J Robot Res 37(4–5):405–420. https://doi.org/10.1177/0278364918770733
36. Thrun S, Burgard W, Fox D (2005) Probabilistic robotics (intelligent robotics and autonomous agents). The MIT Press, Cambridge
37. Widdows D, Cohen T (2015) Reasoning with vectors: a continuous model for fast robust inference. Logic J IGPL/Interest Group Pure Appl Logics 2:141–173
38. Yerxa T, Anderson A, Weiss E (2018) The hyperdimensional stack machine. In: Proceedings of Cognitive Computing, Hannover, pp. 1–2