Cognitive Computation
https://doi.org/10.1007/s12559-024-10258-4

Shift-Equivariant Similarity-Preserving Hypervector Representations of Sequences

Dmitri A. Rachkovskij[1,2]

Received: 14 March 2022 / Accepted: 24 January 2024

Abstract
Hyperdimensional Computing (HDC), also known as Vector-Symbolic Architectures (VSA), is a promising framework for the development of cognitive architectures and artificial intelligence systems, as well as for technical applications and emerging neuromorphic and nanoscale hardware. HDC/VSA operate with hypervectors, i.e., neural-like distributed vector representations of large fixed dimension (usually > 1000). One of the key ingredients of HDC/VSA are the methods for encoding various data types (from numeric scalars and vectors to graphs) by hypervectors. In this paper, we propose an approach for the formation of hypervectors of sequences that provides both an equivariance with respect to the shift of sequences and preserves the similarity of sequences with identical elements at nearby positions. Our methods represent the sequence elements by compositional hypervectors and exploit permutations of hypervectors for representing the order of sequence elements. We experimentally explored the proposed representations using a diverse set of tasks with data in the form of symbolic strings. Although we did not use any features here (hypervector of a sequence was formed just from the hypervectors of its symbols at their positions), the proposed approach demonstrated the performance on a par with the methods that exploit various features, such as subsequences. The proposed techniques were designed for the HDC/VSA model known as Sparse Binary Distributed Representations. However, they can be adapted to hypervectors in formats of other HDC/VSA models, as well as for representing sequences of types other than symbolic strings. Directions for further research are discussed.

Keywords Hyperdimensional computing · Vector symbolic architectures · Brain-like distributed representations · Sequence representation · Similarity preserving transformation · Hypervector permutation

Introduction

Hyperdimensional Computing (HDC [1]), also known as Vector-Symbolic Architectures (VSA [2]), is an approach that has been proposed to combine the advantages of neural-like distributed vector representations and symbolic structured data representations in Artificial Intelligence, Machine Learning, and Pattern Recognition problems. HDC/VSA have demonstrated potential in technical applications and cognitive modelling and are well-suited for implementation in the emerging stochastic hardware (e.g., [3–13] and references therein).

HDC/VSA are one of the few viable proposals [14] for implementing brain-like compositional operations on symbols "on-the-fly", i.e., without training, that appears to be challenging for modern Deep Neural Networks (DNNs) [15]. For another recent non-DNN proposal (that, however, requires learning) see [16]. There is evidence in favor of distributed ("holographic") and sparse representation of information in the brain, e.g., [17–21] and references therein. Brain-like cognitive architectures based on HDC/VSA have been proposed, e.g., in [22, 23]. One of applications of the HDC/VSA-based approach proposed in this paper is cognitive modelling of visual word recognition and similarity in humans [24–27].

HDC/VSA operate with hypervectors (the term proposed in [1]), i.e., brain-like distributed vector representations of large fixed dimension. To be useful in applications (e.g., in various types of similarity search, in linear models for classification, approximation, etc.), hypervectors must be formed to be similar for similar data.

✉ Dmitri A. Rachkovskij
dmitri.rachkovskij@ltu.se; dar@infrm.kiev.ua

1 Luleå University of Technology, Luleå 97187, Sweden

2 International Research and Training Center for Information Technologies and Systems, Kiev, 03680, Ukraine

Published online: 12 March 2024

Methods for obtaining hypervectors for data of various types have been proposed, from numeric scalars and vectors to graphs, e.g., [28–37].

A widespread data type is sequences and, in particular, symbol strings. Sequences and strings are used to represent genome and proteome, signals, textual data, computer logs, etc. Applications that benefit from sequential data representation include bioinformatics, text retrieval and near-duplicate detection, spam identification, virus and intrusion detection, spell checking, signal processing, speech and handwriting recognition, error correction, and many others (e.g., [38–42] and references therein).

The methods of similarity search, clustering, classification, etc., require an assessment of sequence similarity. Formation of hypervector representations that reflect similarity of sequences opens up the possibility of using a large arsenal of methods developed specifically for vectors. These are methods of statistical pattern recognition, linear and nonlinear methods of classification and approximation, index structures for fast similarity search, selection of informative features, and others.

There are several techniques for the representation of sequences with hypervectors. However, most of them do not satisfy either the requirement of equivariance (see "Equivariance of Hypervectors with Respect to Sequence Shift" section) with respect to the sequence shift or the requirement of preserving the similarity of sequences with identical elements at nearby positions (see "Related Work" section). In this paper, we propose an approach for hypervector representation of symbol sequences that satisfies these two requirements. Our methods are based on the use of hypervector permutations to represent the order of sequence elements and were developed for the HDC/VSA model of Sparse Binary Distribution Representations [43, 44] (SBDR). However, the proposed approach can be adapted for hypervector formats of other HDC/VSA models, as well as for representing sequences of other types.

The main contributions of this paper are as follows:

1. Permutation-based hypervector representation of sequences that is shift-equivariant and preserves the similarity of sequences with the same elements at nearby positions.
2. Measures of hypervector similarity of sequences.
3. Measures of symbolic similarity of sequences that approximate the proposed hypervector similarity measures.
4. Experimental study of the proposed hypervector representations of sequences and similarity measures in several diverse tasks: similarity search (spellchecking), classification (splice junction recognition in genes and protein secondary structure prediction), and cognitive modelling (modeling humans' restrictions on the perception of word similarity and the visual similarity of words).

Background and Basic Notions

Hyperdimensional Computing

In various HDC/VSA models, hypervector (HV) components have a different format. For example, they can be real numbers from the Gaussian distribution (the HRR model [45]) or binary values from {0,1} (the BSC model [46] and the SBDR model). Data HVs are formed from the hypervectors of the data elements, usually without changing the HV dimensionality. For example, for elements-symbols their hypervectors are i.i.d. randomly generated vectors of high dimension D, commonly $D > 1000$. Such random HVs are considered dissimilar. The similarity of HVs is usually measured based on their (normalized) dot product. In a particular task, the same data object is represented by its fixed HV.

A set of data objects (e.g., a set of symbols) is represented by the "superposition" of their HVs, for instance, by component-wise addition for real-valued HVs, or by addition followed by thresholding for binary HVs. Superposition does not preserve information about the order or grouping of the objects. The superimposed HVs of similar sets are similar.

To represent a sequence of data objects, their HVs are modified in a special way. For instance, for a hypervector representation of a symbol at some position, the HV of that position ("role") is "bound" to the HV of the symbol ("filler"). Binding can be performed, e.g., by component-wise conjunction (in SBDR) or by XOR (in BSC) for binary HVs, or by cyclic convolution for real-valued HVs (in HRR). This type of binding is called "multiplicative" binding. In another, "permutative" binding type, a role is represented not by a HV, but by a (random) permutation of dimension D, fixed for the particular role, which is applied to the filler HV. A hypervector resulting from binding contains information about the HVs from which it is formed, i.e., about the role and the filler. Binding operation distributes over superposition operation.

Most of the binding operations produce dissimilar HVs for the case when dissimilar filler HVs are bound with the same role, or when the same filler HV is bound with dissimilar roles. "Dissimilar" means that the similarity value is of the order of that for random HVs. For the bound HVs to be similar, both the HVs of the roles as well as the HVs of the fillers should be similar.

The obtained bound HVs are then superimposed. The resulting HV contains information about the bound HVs in superposition, and the bound HVs, in their turn, contain information about their respective constituents. For example, the HV of a symbol string is formed as a superposition of HVs that result from binding HVs of its symbols and

HVs of their positions in the string. Known schemes for hypervector representation of strings are given in "Related Work" section, and those proposed in this work appear in "Method" section.

This paper uses the HDC/VSA model of SBDR. In SBDR, binary hypervectors are used with a small number of $M << D$ (randomly placed) 1-components, the rest of the components are 0s. Superposition is performed by component-wise disjunction. Though the multiplicative binding procedures exist for SBDR [43], in this paper we only use permutative binding.

Symbol Sequences and their Similarity Measures

We will consider sequences of symbols from a finite alphabet. The symbol (sequence element) x at position i is denoted as x_i. E.g., a_0 denotes a at the beginning of the string (at the initial position), a_{-1} is the same symbol shifted one position left, b_3 is the symbol b shifted 3 positions right, and so on. If a symbol is specified without an index, it is at the initial position: $x \equiv x_0$.

We denote by $x_i y_j \ldots z_k$ the sequence of symbols $x, y, \ldots, z$ at positions, respectively, $i, j, \ldots, k$, e.g., $b_3 c_1 a_4 a_{-3}$. A symbol string (symbols at consecutive positions) is denoted as $x_i y_{i+1} \ldots z_{i+k} \equiv xy \ldots z_i$, e.g., $c_1 b_2 c_3 a_4 \equiv cbca_1 \equiv (cbca)_1$. A string without an index is at its initial position, e.g., $cbca \equiv cbca_0 \equiv c_0 b_1 c_2 a_3$.

Various similarity measures are used for strings [39]. The Hamming distance $\mathrm{dist}_{\mathrm{Ham}}$ is equal to the number of non-matching symbols at the same positions ($\mathrm{sim}_{\mathrm{Ham}}$ is defined as the number of matching symbols). Hamming measures capture the intuitive idea of string similarity: the similarity of a string to itself is greater than to other strings, and the more is the number of mismatchings, the less similar strings are, e.g., $\mathrm{sim}_{\mathrm{Ham}}(cbca_0, cbca_0) > \mathrm{sim}_{\mathrm{Ham}}(cbca_0, cbcb_0) > \mathrm{sim}_{\mathrm{Ham}}(cbca_0, cbab_0)$.

Hamming similarity can be extended to strings of different lengths by augmenting a shorter string with special symbols. One can also compare strings at different positions, for example, $cbca_0$ and $cbca_1$, by representing them as $c_0 b_1 c_2 a_3 \$_4$ and $\$_0 c_1 b_2 c_3 a_4$, where \$ is a special symbol that does not belong to the alphabet of string symbols. The last example, however, breaks the intuition of string similarity, since $\mathrm{sim}_{\mathrm{Ham}}(c_0 b_1 c_2 a_3 \$_4, \$_0 c_1 b_2 c_3 a_4) = 0$, however, these strings seem to be similar to us. This problem is solved by the shift distance [47], defined as the minimum Hamming distance between one string and some cyclic shift of the other string.

An alternative approach to string comparison is the Levenshtein distance $\mathrm{dist}_{\mathrm{Lev}}$ defined as the minimum number of edit operations required to change one string into the other [48]. For $\mathrm{dist}_{\mathrm{Lev}}$, edit operations are symbol insertion, deletion, and substitution. The complexity of calculating $\mathrm{dist}_{\mathrm{Lev}}$ (by dynamic programming) is quadratic of the string length. $\mathrm{dist}_{\mathrm{Lev}}$ is widely used in practice, so methods of speeding up its estimation and usage in similarity search are a direction of intensive research [38, 39, 42].

Alignment-free sequence comparison methods [49] do not use dynamic programming to "align" the whole strings (i.e., to find a match between all symbols in two strings) and their computational complexity is sub-quadratic. The methods are based on n-gram frequencies, the length of common substrings, the alignment of substrings, the use of words with some symbol gaps, etc.

Equivariance of Hypervectors with Respect to Sequence Shift

Let x be an object (input), F be a function performing a representation, $F(x)$ be the result of x representation. F is equivariant with respect to transformations T, S if [50] $F(S(x)) = T(F(x))$. Transformations T, S can be different. If T is the identity transformation, F is invariant with respect to S.

We consider hypervector representations of sequences. Let us represent the sequence x as a HV by applying some function (algorithm) $F(x)$. Then shift x to another position, denote this transformation by $S(x)$. The hypervector of the shifted sequence is obtained as $F(S(x))$. The representation function F equivariant with respect to $S(x)$ must ensure $F(S(x)) = T(F(x))$, where T is some transformation of the hypervector $F(x)$. In other words, the hypervector of the shifted sequence can be obtained not only by transforming this sequence into a hypervector, but just by transforming the HV of the unshifted sequence. In the context of brain studies, this can be considered as mental transformation or mental imagery [51, 52]. Please see "Discussion" section for further discussion of equivariance.

Hypervectors corresponding to symbols/sequences will be denoted by the corresponding bold letters. For example, $F(a_0) = \mathbf{a}_0$, $F(cbca_0) = \mathbf{cbca}_0$, $F(cbca_s) = \mathbf{cbca}_s$. We denote the shift of symbols by s positions by S_s: $S_1(a_0) = a_1$, $S_{-1}(abc) = S_{-1}(abc_0) = abc_{-1} \equiv a_{-1} b_0 c_1$. Let T_s denote the hypervector transform corresponding to S_s. To ensure equivariance, the following must be true: $F(S_s(x)) = T_s(F(x)) = \mathbf{x}_s$ (x is a symbol or sequence). For instanse, for specific symbols or strings: $F(S_1(a_0)) = \mathbf{a}_1 = T_1(F(a_0))$, $F(S_2(abc_0)) = T_2(F(abc_0)) = \mathbf{abc}_2$, $F(S_{-2}(abc_4)) = T_{-2}(F(abc_4)) = \mathbf{abc}_2$, etc. In "Permutative Binding with Position" section, hypervector representations of sequences are shown that are shift-equivariant and use a permutation as T.

Related Work

As mentioned in "Hyperdimensional Computing" section, in the HDC/VSA-based methods for representing sequences, each element of a sequence is associated with a hypervector. For symbol strings, symbols are considered dissimilar and so they are assigned randomly generated (and thereafter fixed) hypervectors in the format of the HDC/VSA model being used. To represent sequence elements at their positions, element HVs are modified in various ways. In [53], the following modifications were identified: multiplicative binding with the position HV, multiplicative binding with the HVs of other (e.g., context) elements, and binding the element HV with its position by permutation. *N*-gram representations are also used. Below, we review some of these approaches in more detail. Let us show that the hypervectors formed by these approaches either do not preserve the similarity of sequences with identical elements at nearby positions or are not shift-equivariant.

Multiplicative Binding with Position

In [54], it was proposed to bind the hypervectors of symbols with the hypervectors of their positions by a multiplicative binding operation ("Hyperdimensional Computing" section). Thus, the HV of the sequence $x_i y_j \ldots z_k$ is formed as $\mathbf{x}_i\mathbf{y}_j \ldots \mathbf{z}_k = F(x_i y_j \ldots z_k) = \mathbf{x} \otimes \mathbf{pos}_i \oplus \mathbf{y} \otimes \mathbf{pos}_j \oplus \ldots \oplus \mathbf{z} \otimes \mathbf{pos}_k$, where $\mathbf{pos}_k$ is the HV of the k-th position, $\otimes$ is the binding operation, $\oplus$ is the superposition operation. For instance, for the string *abc*, the hypervector is formed as $\mathbf{abc} = \mathbf{a} \otimes \mathbf{pos}_0 \oplus \mathbf{b} \otimes \mathbf{pos}_1 \oplus \mathbf{c} \otimes \mathbf{pos}_2$. Such a representation was also considered in [45] and was applied, e.g., in [25, 55]. I.i.d. random hypervectors for positions were used. This representation does not preserve the similarity of symbol hypervectors at nearby positions and is not shift-equivariant.

The following approach allows obtaining shift-equivariance. Some multiplicative binding operations allow recursive binding of a hypervector to itself [45], e.g.:

$$\mathbf{pos}^j = \mathbf{pos} \otimes \ldots_{j\,\text{times}} \ldots \otimes \mathbf{pos}.$$

The representation of the sequence in the form $\mathbf{x}_i\mathbf{y}_j \ldots \mathbf{z}_k = F(x_i y_j \ldots z_k) = \mathbf{x} \otimes \mathbf{pos}^i \oplus \mathbf{y} \otimes \mathbf{pos}^j \oplus \ldots \oplus \mathbf{z} \otimes \mathbf{pos}^k$ allows obtaining the hypervector of the shifted sequence as (using the distributivity of the binding operation over the superposition):

$$\begin{aligned} F(S_s(x_i y_j \ldots z_k)) &= F(x_{i+s} y_{j+s} \ldots z_{k+s}) \\ &= \mathbf{pos}^{i+s} \otimes \mathbf{x}_0 \oplus \mathbf{pos}^{j+s} \otimes \mathbf{y}_0 \oplus \ldots \oplus \mathbf{pos}^{k+s} \otimes \mathbf{z}_0 \\ &= \mathbf{pos}^s \otimes (\mathbf{pos}^i \otimes \mathbf{x} \oplus \mathbf{pos}^j \otimes \mathbf{y} \oplus \ldots \oplus \mathbf{pos}^k \otimes \mathbf{z}) \\ &= T_s(F(x_i y_j \ldots z_k)). \end{aligned} \tag{1}$$

Thus, such a hypervector of a string is equivariant with respect to the string shift for $T_s = \mathbf{pos}^s \otimes$. However, this HV does not preserve the similarity of a symbol at nearby positions, since the position hypervectors are not similar and therefore $\mathbf{pos}^i \otimes \mathbf{x}$ is not similar to $\mathbf{pos}^j \otimes \mathbf{x}$ for $i \neq j$. The MBAT approach [56, 57] has similar properties, however, the position binding is performed by multiplying by a random orthonormal position matrix.

Multiplicative Binding with Correlated Position Hypervectors

As mentioned in [53, 58], if the position hypervectors are similar (correlated) for nearby positions, the hypervector representation preserves the similarity of the symbol at different nearby positions. The binding with correlated roles represented by correlated random matrices was proposed in [59]. We are not aware of transformations that ensure shift-equivariance of such string hypervectors.

Based on the ideas of [45], in [60–66] an approach using multiplicative binding is considered. It represents a coordinate value by converting a random hypervector into a complex one using FFT and raises the result component-wise to the fractional power corresponding to the coordinate value. The HV similarity decreases from 1 to 0 when the coordinate increases from 0 to 1. It could be adapted to the representation of strings by associating positions with small coordinate changes and ensures equivariance (mathematically, at least). However, it works with real-valued hypervectors and does not apply to binary hypervectors, and requires expensive forward and inverse FFT.

Permutative Binding with Position

Using permutations of hypervector components to represent the order of sequence elements has been proposed in [1, 67]. The hypervector of the sequence $x_i y_j \ldots z_k$ is formed as $\mathbf{x}_i\mathbf{y}_j \ldots \mathbf{z}_k = F(x_i y_j \ldots z_k) = \mathrm{perm}_i(\mathbf{x}_0) \oplus \mathrm{perm}_j(\mathbf{y}_0) \oplus \ldots \oplus \mathrm{perm}_k(\mathbf{z}_0)$, where perm_k is the permutation corresponding to the k-th position. Similar ideas were considered in [23, 43, 45, 68].

Let $\mathrm{perm}_k = \mathrm{perm}^k$, where $\mathrm{perm}^k(\mathbf{x}) = \mathrm{perm}(\mathrm{perm}(\mathrm{perm}\ldots_{k\,\text{times}} \ldots \mathrm{perm}(\mathbf{x})\ldots))$ is the sequential application of k identical permutations. Here perm is usually a random permutation and $\mathrm{perm}^0(\mathbf{x}) = \mathbf{x}$. For $k < 0$, $\mathrm{perm}^{-|k|}(\mathbf{x})$ denotes the k permutations inverse to perm. This hypervector representation of a sequence is equivariant with respect to the sequence shift :

$$\begin{aligned} F(S_s(x_i y_j \ldots z_k)) &= F(x_{i+s} y_{j+s} \ldots z_{k+s}) \\ &= perm^{i+s}(\mathbf{x}_0) \oplus perm^{j+s}(\mathbf{y}_0) \oplus \ldots \oplus perm^{k+s}(\mathbf{z}_0) \\ &= perm^s(perm^i(\mathbf{x}) \oplus perm^j(\mathbf{y}) \oplus \ldots \oplus perm^k(\mathbf{z})) \\ &= T_s(F(x_i y_j \ldots z_k)). \end{aligned} \tag{2}$$

However, such a representation does not preserve the similarity of the same symbols at nearby positions, since permutation does not preserve the similarity of the permuted hypervector with the original one.

In [69], the representation of a word was formed from the hypervectors of its letters cyclically shifted by the number of positions corresponding to the letter position in the word. In addition, to preserve the similarity with the words containing the same letters in a different order, the original hypervectors of letters were superimposed into the final hypervector of the word. However, shifting this hypervector would give the hypervector different from that obtained by superimposing the initial letter hypervectors with the hypervectors of the shifted word letters at their positions.

Binding by Partial Permutations

To preserve the similarity of hypervectors when using permutations, [70] proposed to use partial (correlated) permutations. Let us apply this approach to symbol sequences. Symbols are represented by random sparse binary HVs $\mathbf{x}$. The HV of a symbol at position i is formed as follows. Let R ("similarity radius") be an integer. The $\mathbf{x}$ is permuted $\lfloor i/R \rfloor$ times. Then we additionally permute a part of 1-components of the resulting HV, the part being equal to $i/R - \lfloor i/R \rfloor$. The rest of the 1-components coincide with the 1-components of the HV at the position $R\lfloor i/R \rfloor$. HVs of all string symbols obtained in this way are superimposed (by component-wise disjunction). For the HV of a symbol at positions i,j, this method approximates the linearly decreasing similarity characteristic $1 - |i-j|/R$ for $|i-j| < R$. For $|i-j| \geq R$, the similarity is close to 0 (corresponds to the similarity of random hypervectors). Such a decreasing similarity is also observed for the HV of a sequence.

When forming the sequence HV, since all the sequence symbols are at different positions, their HVs are permuted in different ways. Therefore, when shifting the string, the HV of each symbol must be permuted differently, taking into account the current position of the symbol. However, we cannot do this, since we have access only to the holistic hypervector of the whole string. Thus, equivariance is not ensured. We are forced to calculate the new positions of symbols in the shifted string $x_{i+s}y_{j+s} \ldots z_{k+s}$ and re-form the hypervector of the sequence at a new position from the scratch: $F(S_s(x_i y_j \ldots z_k)) = F(x_{i+s}y_{j+s} \ldots z_{k+s}) = \mathbf{x}_{i+s} \vee \mathbf{y}_{j+s} \vee \ldots \vee \mathbf{z}_{k+s}$. Shift-equivariance is also absent in [71], where partial permutations of dense hypervectors are used.

Method

To preserve both the equivariance of hypervector representations of sequences with respect to the shift and the similarity of the sequence hypervectors having the same symbols at nearby positions, we propose to form the HVs of symbols as compositional HVs of a specific structure, using random permutation and superposition. We use the SBDR model ("Hyperdimensional Computing" section).

Hypervector Representation of Symbols

To represent the symbol a, we will form its hypervector $\mathbf{a}$ as follows. Let's generate a random ("atomic") HV $\mathbf{e}_{a\,0}$. Let's form other atomic HVs as: $\mathbf{e}_{a\,i} = \mathrm{perm}(\mathbf{e}_{a\,i-1}) = \mathrm{perm}^i(\mathbf{e}_{a\,0})$. Obtain the hypervector of the symbol a at position i (that is, $\mathbf{a}_i = F(a_i)$ for a given value of R) as $\mathbf{a}_i = \mathbf{e}_{a\,i} \vee \mathbf{e}_{a\,i+1} \vee \ldots \vee \mathbf{e}_{a\,i+R-1}$.

Equivariance

For such a hypervector representation, the equivariance with respect to the symbol shift holds if an appropriate permutation is used as the hypervector transformation T. Indeed,

$$\begin{aligned} T_j(F(a_i)) &= \mathrm{perm}^j(\mathbf{a}_i) = \mathrm{perm}^j(\mathbf{e}_{a\,i} \vee \mathbf{e}_{a\,i+1} \vee \ldots \vee \mathbf{e}_{a\,i+R-1}) \\ &= \mathrm{perm}^j(\mathbf{e}_{a\,i}) \vee \mathrm{perm}^j(\mathbf{e}_{a\,i+1}) \vee \ldots \vee \mathrm{perm}^j(\mathbf{e}_{a\,i+R-1}) \\ &= \mathbf{e}_{a\,i+j} \vee \mathbf{e}_{a\,i+j+1} \vee \ldots \vee \mathbf{e}_{a\,i+j+R-1} = \mathbf{a}_{i+j} = F(a_{i+j}) = F(S_j(a_i)). \end{aligned} \tag{3}$$

Similarity

Let us consider hypervectors $\mathbf{a}_i = \mathbf{e}_{a\,i} \vee \mathbf{e}_{a\,i+1} \vee \ldots \vee \mathbf{e}_{a\,i+R-1}$ and $\mathbf{a}_{i+j} = \mathbf{e}_{a\,i+j} \vee \mathbf{e}_{a\,i+j+1} \vee \ldots \vee \mathbf{e}_{a\,i+j+R-1}$. For $|j| < R$, $\mathbf{a}_i$ and $\mathbf{a}_{i+j}$ have $R-|j|$ coinciding atomic HVs. E.g., for $j > 0$ these are atomic HVs with the indices from $i+j$ to $i+R-1$ (the last atomic HVs from $\mathbf{a}_i$ and the first atomic HVs from $\mathbf{a}_{i+j}$; for $j < 0$, the opposite is true). For $|j| \geq R$, $\mathbf{a}_i$ and $\mathbf{a}_{i+j}$ have no coinciding atomic HVs.

For atomic hypervectors $\mathbf{e}$ with the number of 1-components $|\mathbf{e}| = m << D$, their intersection is small with high probability. For the case without the intersection of atomic HVs, the similarity of symbol hypervectors $\mathbf{a}_i$ and $\mathbf{a}_{i+j}$ at different positions inside R is $m(R-|j|)$ (in terms of the number of coinciding 1-components). For the case with the intersection of atomic HVs, the similarity of $\mathbf{a}_i$ and $\mathbf{a}_{i+j}$ inside R may somewhat vary around this value, and there may be similarities between $\mathbf{a}_i$ and $\mathbf{a}_{i+j}$ outside R.

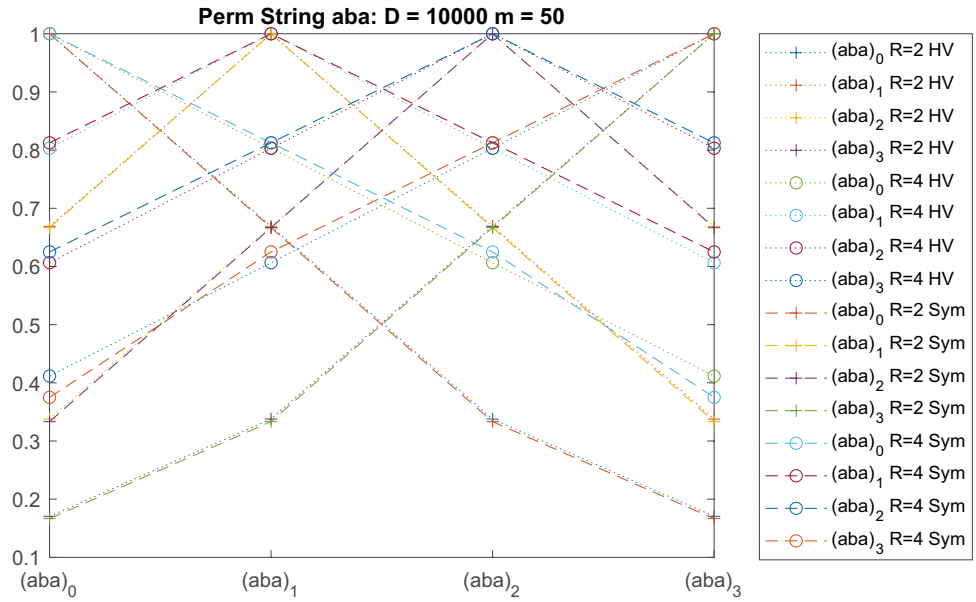

Fig. 1 The hypervector (HV) and the symbolic (Sym) similarity of the string *aba* with itself at different positions 0, 1, 2, 3. The similarity radii $R=2$ and $R=4$. No shifts used when calculating similarity by (4) and (7). Similarity type: cosine

Hypervector Representation and Similarity of Sequences

Hypervectors of various symbols at their positions are formed by the method of "Hypervector Representation of Symbols" section from their randomly generated atomic HVs, using the same permutation. Generally, for a random permutation, some intersection of the HVs of symbols x_i и y_j is possible for any $x \neq y$ and for any i,j. A symbol sequence HV is formed from the symbol HVs using the permutation and superposition operations: $\mathbf{x}_i\mathbf{y}_j \ldots \mathbf{z}_k = F(x_i y_j \ldots z_k) = \mathrm{perm}^i(\mathbf{x}_0) \vee \ldots \vee \mathrm{perm}^k(\mathbf{z}_0)$.

The properties of equivariance and preservation of similarity for hypervectors of symbol sequences can be obtained in the same manner as in "Hypervector Representation of Symbols" and "Permutative Binding with Position" sections. This is achieved due to the distributive property of vector permutation over the superposition operation (the permutation distributivity is also preserved over any component-wise operation on vectors [1]).

Hypervector Similarity of Strings Without Shift

The hypervector similarity is calculated using usual similarity measures of binary vectors. The normalized similarities with the values in [0,1] are given, e.g., by the following measures.

The cosine similarity: $\mathrm{sim}_{\cos} = |\mathbf{a} \wedge \mathbf{b}| \,/\, \mathrm{sqrt}(|\mathbf{a}||\mathbf{b}|) \equiv \langle \mathbf{a}, \mathbf{b} \rangle \,/\, \mathrm{sqrt}(\langle \mathbf{a}, \mathbf{a} \rangle \langle \mathbf{b}, \mathbf{b} \rangle)$, where $|\mathbf{x}| \equiv \langle \mathbf{x}, \mathbf{x} \rangle$ is the number of 1-components in $\mathbf{x}$, $\langle \cdot, \cdot \rangle$ is the dot product. Jaccard: $\mathrm{sim}_{\mathrm{Jac}} = |\mathbf{a} \wedge \mathbf{b}| \,/\, |\mathbf{a} \vee \mathbf{b}| = |\mathbf{a} \wedge \mathbf{b}| \,/\, (|\mathbf{a}| + |\mathbf{b}| - |\mathbf{a} \wedge \mathbf{b}|)$. Simpson: $\mathrm{sim}_{\mathrm{Simp}} = |\mathbf{a} \wedge \mathbf{b}| \,/\, \min(|\mathbf{a}|, |\mathbf{b}|)$.

Let us denote by $\mathrm{sim}_{\mathrm{HV},R,\mathrm{type}}(a,b)$ the measure of hypervector similarity of symbol sequences a,b. HVs are obtained by the method proposed above for a given R value. The "type" stands for, e.g., cos, Jac, Simp, etc. This similarity measure is alignment-free, see "Symbol Sequences and their Similarity Measures" section. Examples of hypervector similarity characteristics for a string at different positions are shown in Fig. 1. The larger value of R provides less steep similarity slopes.

Hypervector Similarity of Strings with the Shift

Strings might have identical substrings outside R. For instance, for $dddabc_0$ and abc_0, the value of $\mathrm{sim}_{\mathrm{HV,R,type}}$ is close to zero for $R \geq 3$. However, if abc_0 is shifted to abc_3, the string abc_3 will match the substring of $dddabc_0$. Let us take into account such cases by calculating the similarity as the maximum value of $\mathrm{sim}_{\mathrm{HV},R,\mathrm{type}}$ for various shifts of one of the sequences:

$$\begin{aligned} \mathrm{sim}_{\mathrm{HV},R,s,\mathrm{type}}(a,b) &\equiv \mathrm{sim}_{\mathrm{HV},R,s,\mathrm{type}}(F(a),F(b)) \equiv \mathrm{sim}_{\mathrm{HV},R,s,\mathrm{type}}(\mathbf{a},\mathbf{b}) \\ &= \max_s \mathrm{sim}_{\mathrm{HV},R,s,\mathrm{type}}(S_s(a),b). \end{aligned} \tag{4}$$

Unless stated otherwise, we assume that the numeric value s specifies the set of shifts from $-s$ to s in steps of 1. For instance, if $s=1$ then $\mathrm{sim}_{\mathrm{HV},R,s,\mathrm{type}}(a,b)$ is the max value of $\mathrm{sim}_{\mathrm{HV},R,\mathrm{type}}(S_s(a),b)$ obtained with shifts $\{-1,0,1\}$ of the sequence a. An example of the resulting similarity characteristics is shown in Fig. 2. Equivariance permits obtaining the HVs of shifted sequences by permuting the sequence HV obtained for a single position. For brevity, if the values of R, s, type are clear, we denote our hypervector similarity measures as $\mathrm{sim}_{\mathrm{HV}}$.

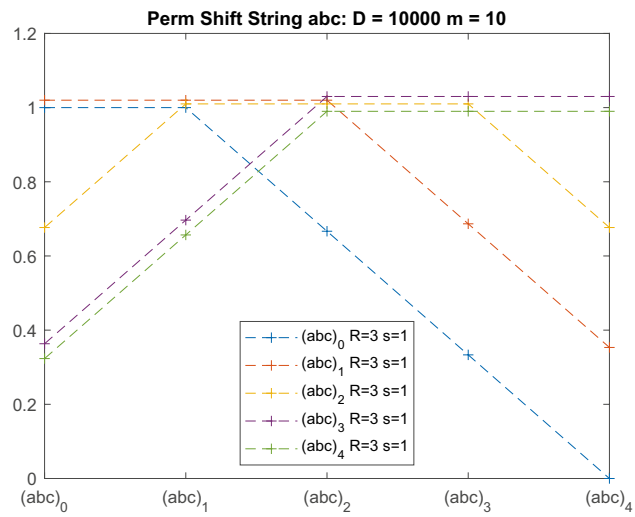

Fig. 2 The values of $\mathrm{sim}_{\mathrm{HV,cos}}$ between the hypervector representation of the string *abc* at different positions, taking into account the shift $s=1$ configuration in (4) (i.e., shifts from $\{-1,0,1\}$). $R=3$

A Symbolic Similarity Measure for Symbol Sequences

Let us introduce a symbolic similarity measure for symbol sequences that is analogous to the proposed $\mathrm{sim}_{\mathrm{HV}}$ but does not use the transformation of strings into hypervectors. We denote: symbol sequences as a, b; an element of sequence x at the position i as x_i; the similarity radius as $R \subset \mathbb{Z}_{\geq 0}$ (a fixed non-negative integer); $\delta_{ij}=|i-j|$;

$$\Delta_{i,R} = \sum_j 1-\delta_{ij}/R \ \text{ if } a_i \in b \wedge \delta_{ij} \leq R|j : b_j = a_i; \quad \Delta_{i,R} = 0 \text{ otherwise.} \tag{5}$$

Then the measure of string similarity, which we call "symbolic overlap" SymOv, is given by

$$\mathrm{sim}_{\mathrm{SymOv},R}(a,b) = \sum_i \Delta_{i,R}. \tag{6}$$

This SymOv similarity is analogous to $|\mathbf{a}\wedge\mathbf{b}|$ for hypervectors of strings a, b. To obtain normalized similarities with the values in [0,1] (analogous to $\mathrm{sim}_{\mathrm{HV}}$ from "Hypervector Similarity of Strings without Shift" section), we define the SymOv-norm of a symbol sequence x as $|x|_R = \mathrm{sim}_{\mathrm{SymOv},R}(x,x)$. Then different types of normalized similarities $\mathrm{sim}_{\mathrm{Sym},R,\mathrm{type}}(a,b)$ are defined analogously to $\mathrm{sim}_{\mathrm{HV},R,\mathrm{type}}(a,b)$.

Taking into account shifts, we obtain:

$$\mathrm{sim}_{\mathrm{Sym},R,s,\mathrm{type}}(a,b) = \max_s \mathrm{sim}_{\mathrm{Sym},R,\mathrm{type}}(S_s(a),b). \tag{7}$$

The values of these similarities would coincide with the (expected) values of hypervector measures, provided that the symbol HVs are superimposed in the sequence HV by addition instead of component disjunction, and $|\mathbf{x}\wedge\mathbf{y}|$ is changed to $\langle \mathbf{x}, \mathbf{y} \rangle$.

Experiments

Experimental evaluations of the proposed approach were carried out in several diverse tasks: spellchecking ("Spellchecking" section), classification of molecular biology data ("Classification of Molecular Biology Data" section), modeling the identification of visual images of words by humans ("Modeling Visual String Identification by Humans" section). In the scope of this paper, the intention of the experiments was to demonstrate the feasibility of the proposed approach to hypervector representation of sequences and its applicability to diverse problem setups.

Spellchecking

For misspelled words, a spellchecker suggests one or more variants of the correct word. We used similarity search for this problem. Dictionary words and misspelled (query) words were transformed to hypervectors by the methods of "Method" section. A specified number of dictionary words with the HVs most similar to the HV of a query word were selected.

As in [72–75], the measure Top-$n=t_n/t$ was used as an indicator of quality or accuracy, where t_n is the number of cases where correct words are contained among the n words of the dictionary most similar to the query, t is the number of queries (i.e., the size of the test set). Two datasets were used: aspell[1] and wikipedia.[2] The tests contain misspellings for some English words and their correct spelling. Our results are obtained with the corncob[3] dictionary containing 58,109 lowercase English words.

Figure 3 shows the aspell Top-n vs R for $n=\{1,10\}$ and their average (Top-mean) for $n=1\ldots10$. Here and thereafter, the dimension of HVs is $D=10{,}000$. $R=1$ corresponds to the lack of similarity between letter HVs at adjacent positions. As R increases, the results improve upto $R=6$–8, then deteriorate slowly.

Our results obtained using $\mathrm{sim}_{\mathrm{HV}}$ and $\mathrm{sim}_{\mathrm{Sym}}$ for various parameters are shown in Table 1. For HVs, means and stds are given (over 50 realizations). The results of Word®, Ispell [75], Aspell [75], and the spellcheckers from [72–74] are also shown. All these spellcheckers work with single words,

[1] http://aspell.net/test/cur/batch0.tab

[2] https://www.dcs.bbk.ac.uk/~ROGER/wikipedia.dat

[3] https://github.com/sibosop/speclib/blob/master/corncob_lowercase.txt

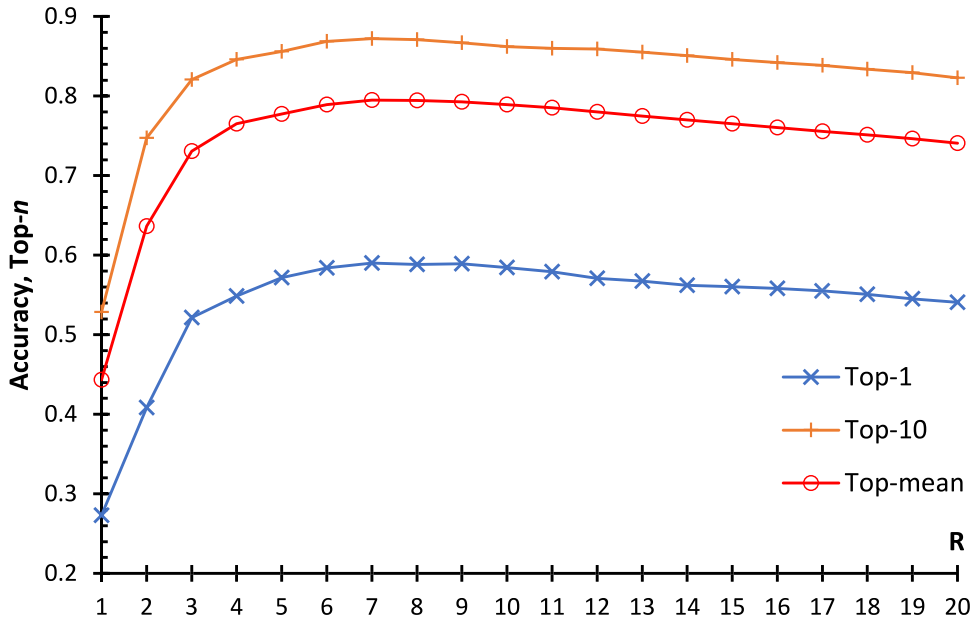

Fig. 3 The Top-1, Top 10, and Top-mean spellchecking accuracy on the aspell dataset vs the similarity radius R. $\mathrm{Sim}_{\mathrm{HV,cos}}$, no shifts in (4)

i.e., do not take into account the adjacent words. However, the results of [72, 73, 75] were obtained using methods specialized for English (using rules, word frequencies, etc.). In contrast to those results, our approach extends naturally to other languages.

Only [74] worked with HV representations and corncob. However, they used the HVs of all 2-grams in the forward direction and in the backward direction, as well as all subsequences (i.e., non-adjacent letters) of two letters in the backward direction. This is in striking contrast to our HV representations, which reflect only the similarity of the same individual letters at nearby positions. Our results are at the level of [74]. Our best results were obtained for $s=0$ (no string shifts, see "Hypervector Similarity of Strings without Shift" section). Increasing s did not lead to a noticeable result change. Note that the similarity search using $\mathrm{dist}_{\mathrm{Lev}}$ and $\mathrm{dist}_{\mathrm{Lev}}$/max (divided by the length of the longer word) produced results that are inferior to ours.

Table 1 Spellchecking accuracy on the aspell and wikipedia tests

Study	Method	Top1 %	Top3 %	Top5 %	Top10 %	Top1 %	Top3 %	Top5 %	Top10 %
		aspell				wikipedia			
[72]	Ispell	36.0	47.7	50.3	51.7	76.0	82.8	83.2	83.4
[72]	Aspell (normal)	56.9	74.4	81.0	87.9	84.7	95.6	97.4	98.5
[72]	Word 97	59.0	69.0	71.0	72.6	89.0	94.3	94.7	95.0
[72]	Word 2003	62.8	74.1	77.2	78.2	92.6	96.1	96.5	96.6
[72]	Deorowicz et al	66.3	79.6	83.6	85.5	94.1	98.3	98.9	99.0
[73]	Mitton	71.1	88.6	91.4	94.4	92.9	97.9	98.6	99.0
[74]	Omelchenko HV	58.6	77.7	82.4	88.9	80.0	92.8	95.7	97.5
Our	Lev	47.8	67.1	73.9	82.2	66.1	81.6	85.7	90.0
Our	Lev/max	54.2	73.1	78.9	85.9	70.9	83.3	86.5	89.7
Our	Sym R = 7	56.7	74.5	78.9	85.1	81.2	94.2	96.4	97.7
Our	Sym R = 7 s = 1	55.9	74.7	78.5	84.7	81.1	94.7	96.1	97.9
Our	HV R = 7 m = 11 std (50)	59.0 0.467	76.5 0.419	81.7 0.408	87.2 0.262	82.8 0.279	93.9 0.134	96.4 0.125	98.0 0.071
Our	HV R = 7 m = 11 s = 1 std (50)	59.4 0.522	76.3 0.482	81.1 0.406	86.4 0.471	84.39 0.234	94.36 0.165	96.61 0.110	97.93 0.0778
Our	HV R = 7 m = 11 s = 2 std (50)	59.2 0.559	75.8 0.449	80.8 0.354	86.3 0.479	83.78 0.27	94.37 0.177	96.55 0.116	96.86 0.0802

Table 2 Accuracy on the splice-junction gene sequences dataset

Study	Method	Total %	EI %	IE %	Neither %
[76]	Hybrid KBANN	–	92.44	91.53	95.38
[77]	C4.5 decision-tree	95.7	–	–	–
[77]	C5.0 rules	95.5	–	–	–
[77]	SLIPPER (rules + AdaBoost)	94.1	–	–	–
[78]	kNN Global Alignment k = 5	93.90	–	–	–
[78]	SVM	97	–	–	–
[79]	C4.5 + Boosting (rule-based)	94.7	96.46	92.81	94.84
[80]	Naive Bayes	94.80	–	–	–
[80]	General Bayesian network (K2)	96.22	–	–	–
[81]	Convolutional NN	96.18	–	–	–
[55]	HDNA (Encoder II)	93.4	96.7	91.5	92.15
Our	Lev kNN k = 27	84.82	88.31	96.75	77.64
Our	Sym kNN R = 1 k = 425	96.71	94.16	98.70	96.98
Our	Sym kNN R = 1 s = 1 k = 170	90.45	83.77	94.81	91.54
Our	Sym kNN R = 2 k = 375	91.24	87.01	97.40	90.33
Our	HV + SVM R = 1 BC = KS = 100.0 opt	**97.03**	**98.05**	**97.40**	**96.37**
Our	HV + SVM R = 1 m = 11 std = 0.289	95.63	93.83	97.00	95.82
Our	HV + SVM R = 1 m = 111 std = 0.326	95.85	93.32	97.01	96.48
Our	HV + Prototypes R = 1 m = 111 std = 0.483	94.09	96.27	98.43	91.06
Our	HV + kNN R = 1 k = 425 m = 11 std = 0.333	96.16	95.22	98.56	95.48

The best results are given in bold

Classification of Molecular Biology Data

Experiments were carried out on two Molecular Biology datasets from [76]. For hypervectors, we used the following classifiers: nearest neighbors *k*NN (mainly with $\text{sim}_{\text{HV,cos}}$), Prototypes, and linear SVM. In Prototypes, class prototypes were obtained by summing the HVs of all training samples from the class; their max similarity with the test HV was used. For SVM, in some cases, the SVM hyperparameters for a single realization of hypervectors were selected by optimization on the training set. The same SVM hyperparameters (or default) were used for multiple HV realizations.

Splice Junction Recognition

The Splice-junction Gene Sequences dataset[4] [76] contains gene sequences for which one needs to recognize the class of splice junctions they correspond to: exon–intron (EI), intron–exon (IE), and no splice (Neither). Each sequence contains 60 nucleotides. The database consists of 3190 samples; 80% of each class was used for training and 20% for testing. Recognition results (accuracy) are shown in Table 2. The results obtained using hypervectors are on a par with the results of other methods from [76–81]. Note that a direct comparison of the results is not fair due to different data partitioning into training and test sets.

The best results were obtained for $R = 1$, $s = 0$. This corresponds to an element-wise comparison of the sequences. We explain this by the fact that the sequences in the database are well-aligned and the recognition result in this problem depends on the presence of certain nucleotides at strictly defined positions. Nevertheless, the introduced hypervector representations and similarity measures demonstrate competitive results for the selected parameters.

Protein Secondary Structure Prediction

The Protein Secondary Structure dataset[5] [76] contains some globular proteins data from [82], and the task is to predict their secondary structure: random-coil, beta-sheet, or alpha-helix. As input data, a window of 13 consecutive amino acids was used, which was shifted over proteins. For each window position and for the amino acid in the middle of the window, the task was to predict what secondary structure it is a part of within the protein. The training/test sets contained 18,105/3520 samples. The prediction results

[4] https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Splice-junction+Gene+Sequences).

[5] https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Protein+Secondary+Structure).

Table 3 Prediction accuracy on the protein secondary structure dataset

Method	Total %	Coil %	Sheet %	Helix %
Backprop 13 amino acids input [82]	62.7	–	–	–
Lev kNN, k = 37 (of 100)	58.72	95.73	21.32	5.614
HV + Prototypes m = 11 R = 1	54.23	50.24	62.02	55.66
HV + SVM R = 1 m = 1 opt	62.78	83.05	45.70	30.08
HV + SVM R = 2 m = 1 opt	**63.21**	83.05	46.29	31.41
HV + SVM R = 3 m = 1 opt	62.47	82.89	45.23	29.55
HV + SVM, R = 1 m = 11 std	62.67 0.0934	84.53 0.142	44.12 0.265	27.53 0.223
HV + SVM, R = 2 m = 11 std	**62.86** 0.0567	82.97 0.0781	45.65 0.1432	30.69 0.1457
HV + kNN k = 28 m = 111 R = 1 std	57.39 0.5261	90.68 0.5101	22.32 1.550	11.64 1.011
HV + kNN k = 28 m = 111 R = 2 std	58.36 0.3682	88.49 0.3277	27.68 1.036	15.73 0.8358

The best results are given in bold

are shown in Table 3. The results of HV and linear SVM for $R = 1$ are at the level of 62.7% [82] obtained by multilayer perceptron for the same experimental design. Using $R = 2$ slightly improved the results obtained with $R = 1$.

Note, that the results obtained on this dataset under this very setup using other methods are inferior to ours (see, e.g., [82]). To improve the results in this and similar tasks, some techniques after [82] used additional information, such as the "similarity" of amino acids, etc. This information can be taken into account in the varied similarity of HVs representing different amino acids; however, this is beyond the scope of this paper (see also Discussion).

The results of this section show that not all string processing tasks benefit from accounting for symbol insertions/deletions (in our approach, regulated by R) and string shifts (regulated by s). For instance, for the Splice-junction dataset, $R > 1$ worsened the results, and for the Secondary-structure dataset, $R = 2$ only slightly improved them. However, we see that the HV representations provide worthy classification results using linear vector classifiers.

The tasks in which the results depend significantly on both parameters R and s are considered in the next section.

Modeling Visual String Identification by Humans

Here we present the results of experiments on the similarity of words using their hypervector representation. The results are compared to those obtained by psycholinguists for human subjects and provided in [24, 26]. Those experiments investigated priming for visual (printed) words in humans.

Modeling Restrictions on the Perception of Word Similarity

In [25], the properties of visual word similarity obtained by psycholinguists in experiments with human subjects have been summarized and classified into 4 types of constraints, i.e., stability (similarity of a string to any other is less than to

Table 4 Constraints on human perception of visual word similarity that are satisfied by various models

Study	Method	Stability	Edge Eff.	Loc TL	Glob TL	Dist TL	Comp TL	Dist RP	Rep RP
[25]	HV BSC Slot coding	✓	–	✓	✓	–	✓	–	–
[25]	HV BSC COB	✓	–	✓	–	–	–	✓	✓
[25]	HV BSC UOB	✓	✓	✓	–	–	–	✓	✓
[25]	HV BSC LCD	✓	✓	✓	–	–	–	✓	–
[25]	Spatial Coding	–	✓	✓	✓	✓	✓	✓	✓
[25]	Seriol	✓	–	✓	–	✓	✓	✓	–
[58]	HV BSC bin 2X	✓	–	✓	–	–	✓	✓	✓
[58]	HV HRR real 2X	✓	–	✓	–	–	✓	✓	✓
[58]	1–Lev/AddedLength	✓	–	✓	✓	–	–	✓	✓
[26]	HV HRR Terminal	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**
Our	minus Lev	✓	–	–	✓	–	✓	✓	✓
Our	Sym cos R = 2 s = 2	✓	–	✓	✓	–	✓	✓	✓
Our	Sym cos R = 3 s = 2	✓	–	✓	–	✓	✓	✓	✓
Our	Sym cos R = 2 s = 2 db	✓	✓	✓	✓	–	✓	✓	✓
Our	Sym cos R = 3 s = 2 db	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**
Our	HV cos R = 2 s = 2 db	✓	✓	✓	✓	–	✓	✓	✓
Our	HV cos R = 3 s = 2 db	✓	✓	✓	✓	✓	✓	✓	?
Our	HV Simp R = 3 s = 2 db	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**

The best results are given in bold

itself); edge effect (the greater importance of the outer letters coincidence vs the inner ones); transposed letter (TL) effects (transposing letters reduces similarity less than replacing them with others); relative position (RP) (breaking the absolute letter order while keeping the relative one still gives effective priming).

Table 4 shows which constraints on human perception of visual word similarity are satisfied in various models. Our results were obtained for $\mathrm{sim}_{\mathrm{Sym}}$ and $\mathrm{sim}_{\mathrm{HV}}$ ($D=10{,}000$, $m=11$, 50 realizations). To reflect the edge effect, we used the "db" option: the HVs were formed in a special way equivalent to the HV representation of strings with doubled first and last letters.

For $s>2$, the results coincided with $s=2$. The best fit to the human constraints was for $R=\{2,3\}$, $s=2$. For $\mathrm{sim}_{\mathrm{Sym,cos}}$ and for $\mathrm{sim}_{\mathrm{HV,Simp}}$ all constraints are satisfied for $R=3$, $s=2$.

For comparison, the results of other models are shown:

Hannagan et al. [25] used the HV representations of the BSC model [46], with the following options. Slot: superposition of HVs obtained by binding HVs of each letter and its (random) position HV. COB: all subsequences of two letters with a position difference of up to 3. UOB: all subsequences of two letters. LCD: a combination of Slot and COB. [25] also tested non-hypervector models: Seriol and Spatial [24].

Cohen et al. [58] used the BSC model and real- and complex-valued HVs of HRR [45]. Position HVs were correlated, and their similarity decreased linearly along the length of a word.

Cox et al. [26] proposed the “terminal-relative” string representation scheme. It used the representation of letters and 2-grams without position, as well as the representation of the positions of letters and 2-grams relative to the terminal letters of the word. This scheme was implemented in the HRR model and met all the constraints from [25].

Modeling the Visual Similarity of Words

In [27], the experimental data on the visual word identification by humans were adapted from [24]. 45 pairs of prime-target strings were obtained, for which there exist the times of human word identification under different types of priming.

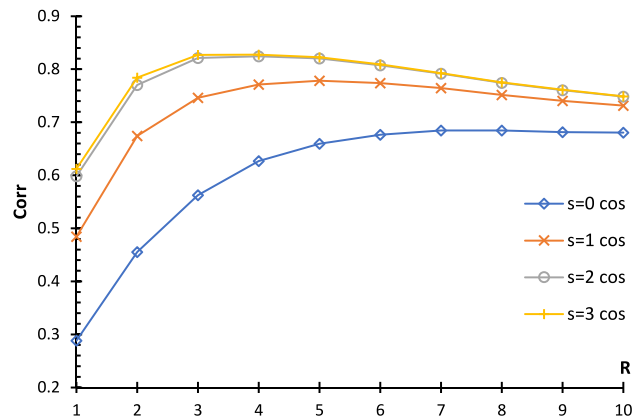

Fig. 4 Mean values of the Pearson correlation coefficient Corr between word hypervector similarity values and the times of the word forward priming vs R. $\mathrm{Sim}_{\mathrm{HV,cos}}$, shift configurations $s=1, 2, 3, 4$ in (4)

Figure 4 shows the average value of the Pearson correlation coefficient Corr (between the $\mathrm{sim}_{\mathrm{HV}}$ values and priming times) vs R for different s ($D=10{,}000$, $m=111$, 50 HV realizations). It can be seen that the value of Corr depends substantially on both R and s. The maximum values were obtained at $R=3$ and $s=2$.

Table 5 shows the Corr between the times of human identification and the values of similarity. The results for HVs were obtained for $R=3$, $s=2$ ($D=10{,}000$, $m=111$, 50 HV realizations) and for similarity measures $\mathrm{sim}_{\mathrm{HV,Jac}}$, $\mathrm{sim}_{\mathrm{HV,cos}}$, $\mathrm{sim}_{\mathrm{HV,Simp}}$. The db option was used. We also provide results for $\mathrm{dist}_{\mathrm{Lev}}$ and $\mathrm{sim}_{\mathrm{Sym,Simp}}$ ($R=3$, $s=2$) without the db option.

The results from [27] are shown as well, where strings were transformed to vectors whose components correspond to certain combinations of letters, with the following variants. Spatial Coding: adapted from [24]. GvH UOB: all subsequences of two letters are used. Kernel UOB (Gappy String kernel): uses counters of all subsequences of two letters within a window. 3-WildCard (gappy kernel): kernel string similarity [27] (all subsequences of two letters are padded with * in all acceptable positions, the vector contains the frequency of each obtained combination of three symbols).

Table 5 The Pearson correlation coefficient Corr between 45 word pairs similarity values and the times of the forward priming

Method	Spatial Coding	GvH UOB	Kernel UOB	3Wild Card	Lev/ max	Sym Simp	HV Jac	HV cos	HV Simp
Corr	0.732	0.673	0.747	0.797	0.834	0.843	0.831	0.822	**0.866**

The best result is given in bold

It can be seen that with the proper parameters, the results of hypervector similarity measures are competitive with other best results, such as $\mathrm{dist}_{\mathrm{Lev}}$ and $\mathrm{sim}_{\mathrm{Sym}}$.

Discussion

The paper proposed a hypervector representation of sequences that is equivariant with respect to sequence shifts and preserves the similarity of identical sequence elements at nearby positions. The case of symbol strings was considered in detail. We exploited a feature-free approach, as our hypervector representations of strings have been formed just from the hypervectors of the symbols at their positions and without using features such as, e.g., *n*-grams. We also proposed a similarity measure of symbol strings that does not use hypervectors but approximates their similarity.

The proposed methods were explored in diverse tasks where strings were used: similarity search in spellchecking, classification of molecular biology data, and modeling of human perception of word similarity. The results obtained were on a par with the results by other methods that, however, additionally use *n*-gram or subsequence representations of strings or some other domain knowledge. We hope that these examples will encourage novel research on other types of tasks and applications.

Other Types of Sequence Elements

Our approach allows using various types of sequence elements, i.e., the data types for which hypervector representations are known can be used. They include numeric scalars or vectors, *n*-grams, other sequences, graphs, etc. Moreover, the proposed methods do not demand sequence elements to be in contiguous positions, as in strings. These modifications may require some method adaptations, such as increasing hypervector dimensionality or/and adjusting parameters and techniques.

Also, our approach can be applied to representing vectors with components that are integers in a fixed range (symbols would correspond to the components of the vector, whereas positions would correspond to the components' values).

Equivariance

The equivariance of representations is a desirable property, at least for the following reasons:

- In the equivariant representations, the information about the transformation $S(x)$ for which the representation is obtained is preserved and available for further processing. For example, a hypervector representation equivariant with respect to a sequence shift preserves information about the position of the sequence. This contrasts with the invariant representation, where such information is lost.
- Ensuring equivariance in hypervector representations opens up the possibility to perform their further equivariance-preserving transformations.
- From an equivariant representation, an invariant one can be obtained. For example, this could be done by superposition of the hypervectors obtained for all the transformations with respect to which invariance is required.
- The system gets the ability to operate with the transformed internal representations of objects, instead of recreating them from the transformed objects (an analogue of "mental transformations" mentioned in "Equivariance of Hypervectors with Respect to Sequence Shift").
- Obtaining the hypervector of the transformed object as $T(F(x))$ is computationally more efficient than as $F(S(x))$, if T is easier to calculate than F and there exists previously obtained object hypervector $F(x)$.
- The absence of computation and energy costs for the execution of $F(S(x))$ is important in case of limited resources, e.g., in edge computing.

We also foresee other interesting effects from equivariant hypervector representations, both in line with DNNs [50] and beyond.

Directions for Future Research

In this paper, the proposed approach for hypervector representation of sequences has been detailed and tested for the case of rather short symbolic strings and for the HDC/VSA model of Sparse Binary Distributed Representations [23, 37, 43]. Areas for further research include the following extensions:

- other HDC/VSA models;
- long sequences; hierarchical sequences;
- other data types (besides sequences);
- other types of equivariance (besides shifts);
- other types of application tasks;
- interplay with DNNs.

Some of these extensions look rather straightforward, some will probably require more research and novel solutions. For example, the proposal for the recursive multiplicative binding based on the approach from this paper was considered in [83].

Concerning further progress in the HDC/VSA field, one promising direction is representing different types of data in a single hypervector [8, 23]. For example, different descriptors for a single image [8] or different modalities of object representation [23]. When using permutative hypervector representations, this would require applying different permutations.

Unlike the formation of distributed vector representations in DNNs, no training is needed to form such hypervector representations in HDC/VSA. As another prospective research topic, let us mention distributed associative memories, along the lines proposed in [84], but for sparse hypervectors [85, 86].

Acknowledgements The author would like to thank the Editor and anonymous reviewers for their fruitful comments and suggestions, Denis Kleyko and Evgeny Osipov for their help and support, and the respective organzations/projects for funding.

Funding Open access funding provided by Lulea University of Technology. This work was supported in part by the Swedish Foundation for Strategic Research (SSF, UKR22-0024), VR SAR Sweden (GU 2022/1963), LTU support grant, as well as by the National Academy of Sciences of Ukraine and the Ministry of Education and Science of Ukraine.

Data Availability The datasets used in this study are available in the respective repositories (the links are provided in the paper text).

Declarations

Ethical Approval This article does not contain any studies with human participants or animals performed by any of the authors.

Conflict of Interest The author Dmitri A. Rachkovskij declares that he has no conflict of interest.

References

1. Kanerva P. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. Cognit Comput. 2009;1(2):139–59.
2. Gayler RW. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In Proc Joint Int Conf Cognit Sci ICCS/ASCS. 2003. p. 133–8.
3. Rahimi A, et al. High-dimensional computing as a nanoscalable paradigm. IEEE Trans Circuits Syst I Reg Papers. 2017;64(9):2508–21.
4. Neubert P, Schubert S, Protzel P. An introduction to hyperdimensional computing for robotics. KI-Kunstliche Intelligenz. 2019;33(4):319–30.
5. Rahimi A, Kanerva P, Benini L, Rabaey JM. Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of ExG signals. Proc of the IEEE. 2019;107(1):123–43.
6. Schlegel K, Neubert P, Protzel P. A comparison of Vector Symbolic Architectures. Artif Intell Rev. 2022;55(6):4523–55.
7. Ge L, Parhi KK. Classification using hyperdimensional computing: A review. IEEE Circ Syst Mag. 2020;20(2):30–47.
8. Neubert P, Schubert S. Hyperdimensional computing as a framework for systematic aggregation of image descriptors. in Proc IEEE/CVF Conf Comp Vis Pat Rec. 2021. p. 16938–47.
9. Hassan E, Halawani Y, Mohammad B, Saleh H. Hyper-Dimensional Computing challenges and opportunities for AI applications. IEEE Access. 2022;10:97651–64.
10. Kleyko D, et al. Vector symbolic architectures as a computing framework for emerging hardware. Proc IEEE. 2022;110(10):1538–71.
11. Neubert P et al. Vector semantic representations as descriptors for visual place recognition, in Proc. Robotics: Science and Systems XVII. 2021;83.1–83.11.
12. Kleyko D, Rachkovskij DA, Osipov E, Rahimi A. A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations. ACM Comput Surv. 2023;55(6):1–40 (Article 130).
13. Kleyko D, Rachkovskij DA, Osipov E, Rahimi A. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. ACM Comput Surv. 2023;55(9): 1–52 (Article 175).
14. Do Q, Hasselmo ME. Neural circuits and symbolic processing. Neurobiol Learn Mem. 2021;186:Article 107552.
15. Greff K, van Steenkiste S, Schmidhuber J. On the binding problem in artificial neural networks. 2020. [Online]. Available: arXiv:2012.05208.
16. Papadimitriou CH, Friederici AD. Bridging the gap between neurons and cognition through assemblies of neurons. Neural Comput. 2022;34(2):291–306.
17. Olshausen BA, Field DJ. Sparse coding of sensory inputs. Curr Opin Neurobiol. 2004;14(4):481–7.
18. Rehn M, Sommer FT. A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields. J Comput Neurosci. 2007;22(2):135–46.
19. Eichenbaum H. Barlow versus Hebb: When is it time to abandon the notion of feature detectors and adopt the cell assembly as the unit of cognition? Neurosci Lett. 2018;680:88–93.
20. Stefanini F, Kushnir L, Jimenez JC, et al. A distributed neural code in the Dentate Gyrus and in CA1. Neuron. 2020;107(4):703-716.e4.
21. Gastaldi C, Schwalger T, De Falco E, Quiroga RQ, Gerstner W. When shared concept cells support associations: Theory of overlapping memory engrams. PLoS Comput Biol. 2021;17(12):e1009691.
22. Eliasmith C, Stewart TC, Choo X, Bekolay T, DeWolf T, Tang Y, Rasmussen D. A Large-scale model of the functioning brain. Science. 2012;338(6111):1202–5.
23. Rachkovskij DA, Kussul EM, Baidyk TN. Building a world model with structure-sensitive sparse binary distributed representations. Biol Inspired Cognit Archit. 2013;3:64–86.
24. Davis CJ. The spatial coding model of visual word identification. Psychol Rev. 2010;117(3):713–58.
25. Hannagan T, Dupoux E, Christophe A. Holographic string encoding. Cognit Sci. 2011;35(1):79–118.
26. Cox GE, Kachergis G, Recchia G, Jones MN. Toward a scalable holographic word-form representation. Behav Res Meth. 2011;43(3):602–15.
27. Hannagan T, Grainger J. Protein analysis meets visual word recognition: A case for string kernels in the brain. Cognit Sci. 2012;36(4):575–606.
28. Kussul EM, Rachkovskij DA, Wunsch DC. The random subspace coarse coding scheme for real-valued vectors, in International Joint Conference on Neural Networks (IJCNN). 1999:1;450–5.
29. Rachkovskij DA, Slipchenko SV, Kussul EM, Baidyk TN. Sparse binary distributed encoding of scalars. J Autom Inf Sci. 2005;37(6):12–23.

30. Rachkovskij DA, Slipchenko SV, Misuno IS, Kussul EM, Baidyk TN. Sparse binary distributed encoding of numeric vectors. J Autom Inf Sci. 2005;37(11):47–61.
31. Kleyko D, Osipov E, Senior A, et al. Holographic graph neuron: A bioinspired architecture for pattern processing. IEEE Trans Neural Netw Learn Syst. 2017;28(6):1250–62.
32. Rachkovskij DA. Formation of similarity-reflecting binary vectors with random binary projections. Cybern Syst Anal. 2015;51(2):313–23.
33. Rachkovskij DA. Estimation of vectors similarity by their randomized binary projections. Cybern Syst Anal. 2015;51(5):808–18.
34. Dasgupta S, Stevens C, Navlakha S. A neural algorithm for a fundamental computing problem. Science. 2017;358(6364):793–6.
35. Osaulenko VM. Expansion of information in the binary autoencoder with random binary weights. Neural Comput. 2021;33(11):3073–101.
36. Rachkovskij DA. Some approaches to analogical mapping with structure sensitive distributed representations. J Exp Theor Artif Intel. 2004;16(3):125–45.
37. Rachkovskij DA, Slipchenko SV. Similarity-based retrieval with structure-sensitive sparse binary distributed representations. Comput Intell. 2012;28(1):106–29.
38. Navarro G. A guided tour to approximate string matching. ACM Comp Surv. 2001;33(1):31–88.
39. Yu M, Li G, Deng D, Feng J. String similarity search and join: A survey. Front Comput Sci. 2016;10(3):399–417.
40. Kussul EM, Kasatkina LM, Rachkovskij DA, Wunsch DC. Application of random threshold neural networks for diagnostics of micro machine tool condition. Int Jt Conf Neural Netw (IJCNN). 1998;1:241–4.
41. Goltsev A, Rachkovskij DA. Combination of the assembly neural network with a perceptron for recognition of handwritten digits arranged in numeral strings. Pattern Recogn. 2005;38(3):315–22.
42. Rachkovskij DA. Index structures for fast similarity search for symbol strings. Cybern Syst Anal. 2019;55(5):860–78.
43. Rachkovskij DA, Kussul EM. Binding and normalization of binary sparse distributed representations by context-dependent thinning. Neural Comput. 2001;13(2):411–52.
44. Kleyko D, Osipov E, Rachkovskij DA. Modification of holographic graph neuron using sparse distributed representations. Procedia Comput Sci. 2016;88:39–45.
45. Plate TA. Holographic reduced representation: distributed representation for cognitive structures. Stanford, CA: Center for the study of language and information; 2003.
46. Kanerva P. Binary spatter-coding of ordered k-tuples, in Proc. 6th Int. Conf. Artif. Neural Netw. von der Malsburg C, von Seelen W, Vorbrüggen JC, Sendhoff B, eds. 1996. p. 869–73.
47. Andoni A, Goldberger A, McGregor A, Porat E. Homomorphic fingerprints under misalignments: Sketching edit and shift distances, in Proc. 45th ACM Sym. Th. Comp. 2013. p. 931–40.
48. Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady. 1966;10(8):707–10.
49. Zielezinski A, et al. Benchmarking of alignment-free sequence comparison methods. Genome Biol. 2019;20:Art. no. 144.
50. Cohen T, Welling M. Group equivariant convolutional networks. in Proc. 33rd Int. Conf. Machine Learn. 2016. p. 2990–9.
51. Pearson J, Naselaris T, Holmes EA, Kosslyn SM. Mental imagery: Functional mechanisms and clinical applications. Trends Cogn Sci. 2015;19(10):590–602.
52. Christophel TB, Cichy RM, Hebart MN, Haynes J-D. Parietal and early visual cortices encode working memory content across mental transformations. Neuroimage. 2015;106:198–206.
53. Sokolov A, Rachkovskij D. Approaches to sequence similarity representation. Int J Inf Theor Appl. 2006;13(3):272–8.
54. Kussul EM, Rachkovskij DA. Multilevel assembly neural architecture and processing of sequences. In: Holden AV, Kryukov VI, editors. Neurocomputers and Attention: Connectionism and Neurocomputers, vol. 2. Manchester and New York: Manchester University Press; 1991. p. 577–90.
55. Imani M, Nassar T, Rahimi A, Rosing T. HDNA: energy-efficient DNA sequencing using hyperdimensional computing. Proc. 2018 IEEE EMBS Int Conf Biomed Health Informatics; 2018. p. 271–4.
56. Gallant SI, Okaywe TW. Representing objects, relations, and sequences. Neural Comput. 2013;25(8):2038–78.
57. Gallant SI. Orthogonal matrices for MBAT Vector Symbolic Architectures, and a "soft" VSA representation for JSON. 2022. [Online]. Available: arXiv:2202.04771.
58. Cohen T, Widdows D, Wahle M, Schvaneveldt R. Orthogonality and orthography: Introducing measured distance into semantic space, in Proc. 7th Int. Conf. on Quantum Interaction, Selected Papers, H. Atmanspacher, E. Haven, K. Kitto, and D. Raine, eds. 2013. p. 34–46.
59. Gallant SI, Culliton PP. Positional binding with distributed representations. Proc. 5th Int. Conf. on Image, Vision and Computin; 2016. p. 108–13.
60. Frady EP, Kent SJ, Kanerva P, Olshausen BA, Sommer FT. Cognitive neural systems for disentangling compositions. Proc. 2nd Int. Conf. Cognit. Computing; 2018. p. 1–3.
61. Komer B, Stewart TC, Voelker AR, Eliasmith C. A neural representation of continuous space using fractional binding. Proc. 41st Ann. Meet. Cog Sci Soc.; 2019. p. 2038–43.
62. Voelker AR, Blouw P, Choo X, Dumont NSY, Stewart TC, Eliasmith C. Simulating and predicting dynamical systems with spatial semantic pointers. Neural Comput. 2021;33(8):2033–67.
63. Frady EP, Kleyko D, Kymn CJ, Olshausen BA, Sommer FT. Computing on functions using randomized vector representations. 2021. [Online]. Available: arXiv: 2109.03429.
64. Frady EP, Kleyko D, Kymn CJ, Olshausen BA, Sommer FT. Computing on functions using randomized vector representations (in brief), in NICE 2022: Neuro-Inspired Computational Elements Conference. 2022. p. 115–22.
65. Schlegel K, Mirus F, Neubert P, Protzel P. Multivariate time series analysis for driving style classification using neural networks and hyperdimensional computing, in IEEE Intelligent Vehicles Symposium (IV). 2021. p. 602–9.
66. Schlegel K, Neubert P, Protzel P. HDC-MiniROCKET: Explicit time encoding in time series classification with hyperdimensional computing, in 2022 International Joint Conference on Neural Networks (IJCNN). 2022. p. 1-8. https://doi.org/10.1109/IJCNN55064.2022.9892158.
67. Sahlgren M, Holst A, Kanerva P. Permutations as a means to encode order in word space. Proc. 30th Annual Meeting of the Cogni Sci Soc.; 2008. p. 1300–5.
68. Kleyko D, Osipov E. On bidirectional transitions between localist and distributed representations: the case of common substrings search using Vector Symbolic Architecture. Procedia Comp Sci. 2014;41:104–13.
69. Kleyko D, Osipov E, Gayler RW. Recognizing permuted words with Vector Symbolic Architectures: A Cambridge test for machines. Procedia Comp Sci. 2016;88:169–75.
70. Kussul EM, Baidyk TN, Wunsch DC, Makeyev O, Martin A. Permutation coding technique for image recognition system. IEEE Trans Neural Netw. 2006;17(6):1566–79.
71. Cohen T, Widdows D. Bringing order to neural word embeddings with embeddings augmented by random permutations (EARP), in Proc. 22nd Conf. Computational Natural Language Learning. 2018, p. 465–75.
72. Deorowicz S, Ciura MG. Correcting spelling errors by modeling their causes. Int J Appl Math Comp Sci. 2005;12(2):275–85.
73. Mitton R. Ordering the suggestions of a spellchecker without using context. Nat Lang Eng. 2009;15(2):173–92.
74. Omelchenko RS. Spellchecker based on distributed representations. Problems in Programming. 2013;(4):35–42. (in Russian)

75. Atkinson K. GNU Aspell. [Online]. Available: http://aspell.net/. Accessed 12 Feb 2024.
76. Dua D, Graff C. UCI Machine Learning Repository Irvine, CA: University of California, School of Information and Computer Science. 2019. [Online]. Available: http://archive.ics.uci.edu/ml. Accessed 12 Feb 2024
77. Cohen W, Singer Y. A simple, fast and efficient rule learner, in Proc. 16th Nat. Conf. Artific. Intell. 1999. p. 335–42.
78. Deshpande M, Karypis G. Evaluation of techniques for classifying biological sequences, in Proc 6th Pacific-Asia Conf Adv Knowl Discov Data Mining. 2002. p. 417–31.
79. Li J, Wong L. Using rules to analyse bio-medical data: A comparison between C4.5 and PCL, in Adv Web-Age Inf Manage. Dong G, Tang C, Wang W, eds. 2003. p. 254–65.
80. Madden M. The performance of Bayesian network classifiers constructed using different techniques, in Proc. 14th Eur. Conf. Machine Learn., Workshop on Probabilistic Graphical Models for Classification. 2003. p. 59–70.
81. Nguyen NG, et al. DNA sequence classification by Convolutional Neural Network. J Biomed Sci Eng. 2016;9(5):280–6.
82. Qian N, Sejnowski TJ. Predicting the secondary structure of globular proteins using neural network models. J Mol Biol. 1988;202(4):865–84.
83. Rachkovskij DA, Kleyko D. Recursive binding for similarity-preserving hypervector representations of sequences, in 2022 International Joint Conference on Neural Networks (IJCNN). 2022. p. 1-8. https://doi.org/10.1109/IJCNN55064.2022.9892462.
84. Steinberg J, Sompolinsky H. Associative memory of structured knowledge. Sci Rep. 2022;12:Article 21808.
85. Vdovychenko R, Tulchinsky V. Sparse distributed memory for sparse distributed data, in Proc. SAI Intelligent Systems Conference (IntelliSys 2022). 2022. p. 74–81.
86. Vdovychenko R, Tulchinsky V. Sparse distributed memory for binary sparse distributed representations, in Proc. 7th International Conference on Machine Learning Technologies (ICMLT 2022). 2022. p. 266–70.