In J. A. de Barros, B. Coecke & E. Pothos (eds.)
Quantum Interaction, 10th International Conference,
QI 2016 (July 20-22, San Francisco), pp. 265-274.
Springer, 2017.

Language Geometry using Random Indexing

Aditya Joshi[1], Johan T. Halseth[2], and Pentti Kanerva[3]

[1] Department of Mathematics, University of California–Berkeley
[2] Department of Computer Science, University of California–Berkeley
[3] Redwood Center for Theoretical Neuroscience

Abstract. Random Indexing is a simple implementation of Random Projections with a wide range of applications. It can solve a variety of problems with good accuracy without introducing much complexity. Here we demonstrate its use for identifying the language of text samples, based on a novel method of encoding letter N-grams into high-dimensional Language Vectors. Further, we show that the method is easily implemented and requires little computational power and space. As proof of the method's statistical validity, we show its success in a language-recognition task. On a difficult data set of 21,000 short sentences from 21 different languages, we achieve 97.4% accuracy, comparable to state-of-the-art methods.

Keywords: N-gram Vector, Language Profile, Vector Symbolic Architecture, Multiply–Add–Permute Algebra

1 Introduction

As humans who communicate through language, we have the fascinating ability to recognize unknown languages in spoken or written form, using simple cues to distinguish one language from another. Some unfamiliar languages, of course, might sound very similar, especially if they come from the same language family, but we are often able to identify the language in question with very high accuracy. This is because embedded within each language are certain features that clearly distinguish one from another, whether it be accent, rhythm, or pitch patterns. The same can be said for written languages, as they all have features that are distinctive. Recognizing the language of a given text is the first step in all sorts of language processing, such as text analysis, categorization, translation and much more.

As popularized by Shannon ([1]), most language models use distributional statistics to explain structural similarities in various specified languages. The traditional method of identifying languages in the absence of dictionaries consists of counting individual letters, letter bigrams, trigrams, tetragrams, etc., and comparing the frequency profiles of different text samples. As a general principle, the more accurate you want your detection method to be, the more data you have to store about the various languages. For example, Google's recently open-sourced program called Chromium Compact Language Detector uses large

language profiles built from enormous corpora of data. As a result, the accuracy of their detection, as seen through large-scale testing and in practice, is near perfect ([2]).

High-dimensional vector models are popular in natural-language processing and are used to capture word meaning from word-use statistics. The vectors are called *semantic vectors* or *context vectors*. Ideally, words with a similar meaning are represented by semantic vectors that are close to each other in the vector space, while dissimilar meanings are represented by semantic vectors far from each other. Latent Semantic Analysis is a well-known model that is explained in detail in [3]. It produces 300-dimensional (more or less) semantic vectors from a singular value decomposition (SVD) of a matrix of word frequencies in a large collection of documents.

An alternative to SVD, based on Random Projections, was proposed by Papadimitriou ([4]) and Kaski ([5]). Random Indexing ([6, 7]) is a simple and effective implementation of the idea. It has been used in ways similar to Mikolov et al.'s Continuous Bag-of-Words Model (KBOW; [8]) and has features similar to Locality-Sensitive Hashing (LSH) but differs from them in its use of high dimensionality and randomness. With the dimensionality in the thousands (e.g., D = 10,000)—referred to as "hyperdimensional"—it is possible to calculate useful representations in a single pass over the dataset with very little computing.

In this paper, we will present a way of doing language detection using Random Indexing, which is fast, highly scalable, and space efficient. We will also present some results regarding the accuracy of the method, even though this will not be the main goal of this paper and should be investigated further.

2 Random Indexing

Random Indexing represents information by projecting data onto vectors in a high-dimensional space. There exist a huge number of different, nearly orthogonal vectors in such a space [9, p. 19]. This lets us combine two such vectors into a new vector using well-defined vector-space operations, while keeping the information of the two with high probability. In our implementation of Random Indexing, we use a variant of the MAP (Multiply, Add, Permute) coding described in [10] to define the vector space. Vectors are initially taken from a D-dimensional space (with $D = 10{,}000$) and have an equal number of randomly placed 1s and -1s. Such vectors are used to represent the basic elements of the system, which in our case are the 26 letters of the Latin alphabet and the (ASCII) Space. These vectors for letters are sometimes referred to as their *Random Labels*.

The binary operations on such vectors are defined as follows. Elementwise *addition* of two vectors A and B, is denoted by $A + B$. Similar, elementwise *multiplication* is denoted by $A * B$. A vector A will be its own multiplicative inverse, $A * A = \mathbf{1}$, where $\mathbf{1}$ is the D-dimensional identity vector consisting of only 1s. The cosine is used to measure the similarity of two vectors. It is defined

as $\cos(A, B) = |A' * B'|$, where A' and B' are the normalized vectors of A and B, respectively, and $|C|$ denotes the sum of the elements in C.

Information from a pair of vectors A and B is stored and utilized in a single vector by exploiting the summation operation. That is, the sum of two separate vectors naturally preserves unique information from each vector because of the mathematical properties of the space. To see this, note that $\cos(A, A) = 1$, while for all $B \neq A$, $\cos(A, B) < 1$. The cosine of two random, unrelated vectors tends to be close to 0. Because of this, the vector B can easily be found in the vector $A + B$: $\cos(B, A + B)$ differs significantly from 0.

For encoding a sequence of vectors, we use a random (but fixed throughout all our computations) *permutation* operation ρ of the vector coordinates. Hence, the sequence A-B-C is encoded as the D-dimensional vector ABC by permuting the first vector twice, permuting the second vector once, taking the third vector as is, and by multiplying the tree: $ABC = \rho(\rho(A)) * \rho(B) * C = \rho\rho A * \rho B * C = \rho^2 A * \rho B * C$. This efficiently distinguishes the sequence A-B-C from, say, A-C-B. This can be seen from looking at their cosine (here c is the normalization factor):

$$
\begin{aligned}
V_1 &= \rho\rho A * \rho B * C \\
V_2 &= \rho\rho A * \rho C * B \\
\implies \cos(V_1, V_2) &= c \cdot |(\rho\rho A * \rho B * C) * (\rho\rho A * \rho C * B)| \\
&= c \cdot |\rho\rho A * \rho\rho A * \rho B * \rho C * C * B)| \\
&= c \cdot |\rho\rho(A * A) * \rho(B * C) * (B * C))| \\
&= c \cdot |\mathbf{1} * \rho(B * C) * (B * C))| \\
&\approx c \cdot 0
\end{aligned}
$$

since a random permutation ρV of a random vector V is uncorrelated to V.

2.1 Making and Comparing of Text Vectors

We use the properties of high-dimensional vectors to extract certain properties of text into a single vector. [11] shows how Random Indexing can be used for representing the contexts in which a word appears in a text, into that word's context vector. We show here how to use a similar strategy for recognizing a text's language by creating and comparing *Text Vectors*: the Text Vector of an unknown text sample is compared for similarity to precomputed Text Vectors of known language samples—the latter are referred to as *Language Vectors*.

Simple language recognition can be done by comparing letter frequencies of a given text to known letter frequencies of languages. Given enough text, a text's letter distribution will approach the letter distribution of the language in which the text was written. The phenomenon is called an "ergodic" process in [1], as borrowed from similar ideas in physics and thermodynamics. This can be generalized to using *letter blocks* of different sizes. By a block of size N, we mean N consecutive letters in the text so that a text of length M would have $M - N + 3$ blocks. When the letters are taken in the order in which they appear in the text, they are referred to as a sequences (of length N) or as N-grams.

As an example, the text "a book" gives rise to the trigrams "a_b", "_bo", "boo", and "ook" (here "_" stands for Space). The frequencies of such letter blocks can be found for a text and compared to known frequencies for different languages. For texts in languages using the Latin alphabet of 26 letters (plus Space), like English, this would lead to keeping track of $27^3 = 19{,}683$ different trigram frequencies. For arbitrary alphabets of L letters, there would be $(L+1)^N$ N-grams to keep track of. These numbers grow quickly as the block size N increases, yet Random Indexing encodes all N-gram frequencies into a single 10,000-dimensional Text Vector.

The Random Indexing approach for doing language recognition is similar. A text's Text Vector is first calculated by running over all the blocks of size N within the text and creating an N-gram Vector for each. An N-gram Vector is created for the sequence of letters as described earlier. As an example, if we encounter the block "rab", its trigram vector is calculated by performing $\rho\rho R * \rho A * B$, where R, A and B are the Random Labels for r, a, and b—they are random D-dimensional vector with half 1s and half -1s, and the same ones are used with all languages and text samples.

A text's Text Vector is now obtained from summing the N-gram Vectors for all the blocks in the text. This is still an D-dimensional vector and can be stored efficiently. Language Vectors are made in exactly the same way, by making Text Vectors from samples of a known language and adding them into a single vector. Determining the language of an unknown text is done by comparing its Text Vector to all the Language Vectors. More precisely, the cosine measure $d_{\cos}$ between a language vector X and an unknown text vector V is defined as follows:

$$d_{\cos}(X, V) = \frac{X \cdot V}{|X||V|} = \frac{\sum_{i=1}^{D} x_i v_i}{\sqrt{\sum_{j=1}^{D} x_j^2 \sum_{k=1}^{D} v_k^2}}$$

If the cosine is high (close to 1), the trigram frequencies of the text are similar to the trigram frequencies of that language and thus, the text is likely to be written in the same language. Hence, the language that yields the highest cosine is chosen as the system's prediction/guess.

2.2 Complexity

The outlined algorithm for Text Vector generation can be implemented efficiently. For generating a vector for an N-gram, $N-1$ vector permutations and multiplications are performed. This takes time $O(N \cdot D)$. Looping over a text of M letters, $O(M)$ N-gram Vectors must be created and added together. This clearly implies an $O(N \cdot D \cdot M)$ implementation. This can be improved to $O(D \cdot M)$ by noting that most of the information needed for creating the N-gram Vector for the next block is already contained in the previous N-gram Vector, and can be retrieved by removing the contribution from the letter that is now no longer in the block.

Say we have the N-gram Vector $A = \rho^{N-1} V_1 * \rho^{N-2} V_2 * \ldots * \rho V_{N-1} * V_N$ for block number i, and now want to find the N-gram Vector B for block $i+1$. We

remove from A the vector $\rho^{N-1}V_1$ by multiplying with its inverse (which is the vector itself), which we can do in $O(D)$ time since ρ^{N-1} is just another (precalculated) permutation. Then we permute the result once using ρ and multiply that with the Letter Vector V_{N+1} for the new letter in the block. This gives us the new N-gram Vector

$$\begin{aligned} B &= \rho(\rho^{N-1}V_1 * A) * V_{N+1} \\ &= \rho(\rho^{N-2}V_2 * \ldots * \rho V_{N-1} * V_N) * V_{N+1} \\ &= \rho^{N-1}V_2 * \ldots * \rho^2 V_{N-1} * \rho V_N * V_{N+1} \end{aligned}$$

and so we can create N-gram Vectors for arbitrary size blocks without adding complexity.

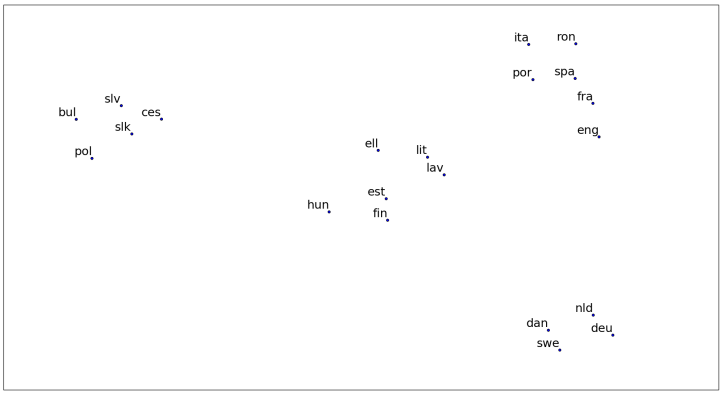

Figure 1: 10,000-dimensional Language Vectors for 21 languages roughly cluster based on the known relations between the languages. The Language Vectors were based on letter trigrams and were projected onto a plane using t-SNE ([12]).

3 Experimental Results

The algorithm outlined above was used to create Language Vectors for 21 languages. Texts for the Language Vectors were taken from the Wortschatz Corpora ([13]) where large numbers of sentences in selected languages can be easily downloaded. Each Language Vector was based on about a million bytes of text. Computing of the Language Vectors corresponds to training the system.

Intuitively, Language Vectors within a language family should be closer to each other than vectors for unrelated languages. Indeed, the hyperdimensional Language Vectors roughly cluster in this manner, as seen in Figure 1.

To get an idea of how well the actual detection algorithm works, we tested the Language Vectors' ability to identify text samples from the Europarl Parallel Corpus, described in [14]. This corpus includes 21 languages with 1,000 samples of each, and each sample is a single sentence.

Table 1 shows the result for N-gram sizes from 1 to 5 ($N = 1$ is the equivalent of comparing letter histograms). With tetragrams we were able to guess the correct language with 97.8% accuracy. Even when incorrect, the system usually chose a language from the same family, as seen from Table 2.

N	Detection success
1	74.9
2	94,0
3	97.3
4	97.8
5	97.3

Table 1: Percentage of sentences correctly identified as a function of N-gram size.

It is worth noting that 10,000 small integers keep track of 14,348,907 possible pentagrams just as easily as 19,683 trigrams. The method should be explored further, as explained in the Future Work section.

The arithmetic (algebra) of the operations with which Text Vectors are made—i.e., permutation, multiplication, and addition, and how they work together—make it possible to analyze the Language Vectors and find out, for example, what letters are most likely to follow "th". In English it would be "e", but what is the next most likely? In Table 3, we answer this question using a learnt Language Vector for English.

4 Details of Implementation

The 21 Language Vectors were "trained" with text from the Leipzig Corpora Collection (website http://corpora.uni-leipzig.de/download.html). The file for each language is about a million bytes and contains 10,000 sentences of news material. Letters outside the 26 in the Latin alphabet were replaced by their Latin equivalents by hand-coding and using the Unidecode 0.04.17 package (https://pypi.python.org/pypi/Unidecode), and sequences of nonletters were treated as a single space. The 21,000 test sentences (1,000 per language) came from the European Parliament Proceedings Parallel Corpus 1996–2011 (http://www.statmt.org/europarl/) and were preprocessed the same way as the training corpus.

The "random," fixed permutation ρ was implemented as a rotate by one coordinate position. This is safe because the vectors themselves are random,

	ell	eng	ita	ces	est	spa	nld	por	lav	lit	ron	pol	fra	bul	deu	dan	fin	hun	swe	slk	slv
ell	987	1	.	.	.	.	3	3	.	.	.	1	.	4	.	.	1	.	.	.	.
eng	2	982	.	4	.	.	1	.	2	.	.	.	6	.	.	1	.	2	.	.	.
ita	.	.	992	.	1	2	.	.	.	.	2	3	.	.	.	.	.	.	.	.	.
ces	1	1	.	940	1	.	.	.	1	1	1	1	.	5	1	.	.	.	.	35	12
est	1	.	.	1	983	.	.	.	3	.	.	.	3	.	1	1	5	1	1	.	.
spa	.	.	6	.	.	946	2	30	8	1	2	.	5	.	.	.	.	.	.	.	.
nld	.	1	.	.	.	.	980	1	.	.	2	1	.	.	5	9	.	.	1	.	.
por	.	1	2	.	.	1	1	991	.	.	.	.	3	1	.	.	.	.	.	.	.
lav	2	.	.	1	.	.	.	2	963	26	.	2	.	2	.	1	.	.	.	1	.
lit	2	.	1	2	1	1	.	2	18	969	.	.	1	.	.	.	.	.	.	1	2
ron	.	.	1	.	.	1	.	2	.	1	987	2	4	2	.	.	.	.	.	.	.
pol	2	1	.	3	1	.	.	.	.	.	.	984	.	4	.	.	.	.	.	4	1
fra	3	.	2	.	.	4	2	1	1	2	1	.	982	.	.	1	.	.	.	1	.
bul	1	.	.	7	.	.	4	.	.	.	.	.	.	984	.	.	.	.	.	3	1
deu	.	2	1	1	.	.	3	.	.	.	.	.	3	.	985	4	.	.	1	.	.
dan	.	2	.	.	.	.	9	.	.	.	.	.	2	.	.	974	.	.	13	.	.
fin	.	.	.	.	4	.	2	.	1	.	.	.	.	.	.	.	993	.	.	.	.
hun	.	.	.	.	.	.	6	1	1	1	.	.	.	.	.	2	.	989	.	.	.
swe	.	1	.	.	.	1	5	.	.	.	4	.	1	.	4	10	.	.	974	.	.
slk	2	.	.	72	.	.	1	.	2	1	4	18	.	6	1	.	.	.	.	881	12
slv	1	.	.	5	2	.	.	1	.	.	1	.	.	6	1	1	.	.	.	.	982

LEGEND: bul = Bulgarian, ces = Czech, dan = Danish, deu = German, ell = Greek, eng = English, est = Estonian, fin = Finnish, fra = French, hun = Hungarian, ita = Italian, lav = Latvian, lit = Lithuanian, nld = Dutch, pol = Polish, por = Portuguese, ron = Romanian, slk = Slovak, slv = Slovene, spa = Spanish, swe = Swedish.

Table 2: The confusion matrix of language detection using 10,000-dimensional Language Vectors based on letter trigrams. Each row corresponds to the correct label and each column is the predicted label for the Europarl corpus detection test. The entry (i, j) is the number of sentences (out of a 1,000) that language j was guessed for language i. A high value diagonal shows the very high accuracy.

only one permutation is needed, and the permutation is iterated a few times at most (much fewer than 10,000).

The experiment was programmed in Python and run on a laptop computer. The following run-time statistics are from a 64-bit, 2.70GHz (100MHz clock) Intel processor, 4 cores and 32GB of 1600 MHz memory (total). Computing a 10,000-dimensional Language Vector from a million bytes of text takes 14.5 seconds. Computing the 10,000-dimensional Text Vectors for the 21,000 test sentences and comparing them to the 21 Language Vectors, to make the confusion matrix, takes 2 minutes. The run time for a round of experiments to make Table starting with a random seed is just over 7 minutes.

Letter	Distance
e	0.31
_	0.063
a	0.049
i	0.024
r	0.024
o	0.018

Table 3: Using the vector operations of multiplication and inverse permutations, and noting that the multiplicative inverse of a random vector with only 1s and −1s is itself, we find the most likely letter to follow the bigram "th", knowing the answer is encoded in an English Vector. As expected intuitively, the result shows that "e" is the most likely letter. Additionally, we have easily accessible information about the second most likely and so on. We show the top 6. (Note that _ is (ASCII) space.)

5 Discussion

Computing with high-dimensional random vectors is the larger issue addressed by this paper: what are the operations on the vectors, what is their algebra, and what kinds of algorithms the algebra favors? Language identification provides us with an easily understood example of the concepts involved.

The addition and multiplication operations on the vectors form an algebraic structure that approximates a field, which is further complemented by a permutation operation that distributes over both addition and multiplication. These operations constitute a kind of Multiply–Add–Permute (MAP) algebra ([15]) that seems particularly suited for modeling human cognition and language.

This style of computing goes back to Hinton's Reduced Representation which emphasizes the need to represent sets and their elements with vectors of equal width ([16]), and to Smolensky's Tensor Product Variable Binding which allows a set of variable–value pairs to be encoded and superposed in a higher-order tensor from which the individual constituents can be extracted ([17]). These two ideas are brought together in Plate's Holographic Reduced Representation (HRR; [18, 19]), of which the present system is a special case. The idea is to work in a *closed* system—namely, that the outputs of addition, multiplication and permutation have the same dimensionality (and statistical distribution) as the inputs. The term Vector Symbolic Architecture (VSA; [20]) refers to systems of this kind.

VSA systems use either multiplication or permutation for variable binding because they are invertible and they distributes over addition. Here we have encoded N-grams using both. First the letters are bound to their positions within an N-gram with permutations and then the position-encoded letters are "bound" to each other with multiplication—this latter "binding" is a more general mapping because it is not between variables and their values. When the N-gram Vectors for a given text are superposed with vector addition, we get an N-grams

profile that can be compared to profiles of other text samples (see Fig. 1 and Table 2).

The example of Table 3 is more subtle, where we query a Language Vector for the letter that appeared most often after "th". The solution can be understood in terms of the vector algebra that makes use of both the inverse permutation and the inverse multiplication. This kind of representation vaguely resembles quantum superposition that allows all the superposed vectors to be operated on in parallel and the results to be extracted with appropriate inverse operations. The simplicity of the algorithm is worth pointing out.

6 Future Work

Many adjustments can be made to improve the efficacy of Random Indexing on language detection. The results of this paper are based mainly on letter trigrams. However, it is a simple matter to add into the Text Vectors single-letter frequencies and bigrams, for example. Also, the vector dimensionality can be reduced to several thousands without markedly affecting the results. Early experiments suggest that this method works well with encoding language information in multilingual texts, which is often much more difficult to do.

Because of the generality of Random Indexing on texts, any time series with a well-defined "alphabet" can be encoded using this scheme. In this way, we propose that our method can be used to do language detection in speech data, addressing our original problem.

7 Conclusion

We have described the use of Random Indexing to language identification. Random Indexing has been used in the study of semantic vectors since 2000 ([6, 7]), and for encoding problems in graph theory ([10]), but only now for identifying source materials. It is based on simple operations on high-dimensional random vectors: on Random Labels with 0-mean components that allow weak signals to rise above noise as the data accumulate. The algorithm works in a single pass, in linear time, with limited memory, and thus is inherently scalable, and it produces vectors that are amenable to further analysis. The experiments reported in this paper were an easy task for a laptop computer.

Acknowledgments: We thank Professor Bruno Olshausen for providing the setting for this work in his class on Neural Computation, and two anonymous reviewers for their comments that helped us improve the paper. Pentti Kanerva's work was supported by Systems On Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

References

[1] C.E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* (1948).

[2] M. McCandless. *Accuracy and performance of Google's Compact Language Detector*. 2011. URL: `http://blog.mikemccandless.com/2011/10/accuracy-and-performance-of-googles.html`.

[3] T. Landauer and S. Dumais. "A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge". In: *Psychology Review* 104.2 (1997), pp. 211–240.

[4] C.H. Papadimitriou et al. "Latent Semantic Indexing: A probabilistic analysis". In: *Proc. 17th ACM Symp. on the Principles of Database Systems* (1998), pp. 159–168.

[5] S. Kaski. "Dimensionality reduction by random mapping: Fast similarity computation for clustering". In: *Proc. IJCNN'98, International Joint Converence on Neural Networks* 1 (1998), pp. 413–418.

[6] P. Kanerva, J. Kristoferson, and A Holst. "Random Indexing of text samples for Latent Semantic Analysis". In: *Proc. 22nd Annual Conference of the Cognitive Science Society*. Ed. by L.R. Gleitman and A.K Josh. 2000, p. 1036.

[7] M. Sahlgren. "An introduction to random indexing". In: *Methods and Applications of Semantic Indexing Workshop at the 7th international conference on Terminology and Knowledge Engineering* (2005).

[8] T. Mikolov et al. "Efficient estimation of word representations in vector space". In: (2013). arXiv:1301.3781v3 [cs.CL] 7 Sep 2013, 12 pp.

[9] P. Kanerva. *Sparse Distributed Memory*. MIT Press, 1988.

[10] S.D. Levy and R.W. Gayler. "Lateral inhibition in a fully distributed connectionist architecture". In: *Proceedings of the Ninth International Conference on Cognitive Modeling* (2009).

[11] P. Kanerva. "Computing with 10,000-bit words". In: *Proc. 52nd Annual Allerton Conference on Communication, Control, and Computing* (2014).

[12] L. van der Maaten. "Visualizing high-dimensional data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[13] U. Quastoff, M. Richter, and C. Biemann. "Corpus portal for search in monolingual corpora". In: *Proceedings of the fifth international conference on Language Resources and Evaluation, LREC* (2006), pp. 1799–1802.

[14] S. Nakatani. *langdetect is updated(added profiles of Estonian / Lithuanian / Latvian / Slovene, and so on.* http://shuyo.wordpress.com/2011/09/29/langdetect-is-updatedadded-profiles-of-estonian-lithuanian-latvian-slovene-and-so-on/. [Online; accessed 16-December-2014].

[15] R. W. Gayler. "Multiplicative binding, representation operators, and analogy". In: *Advances in Analogy Research*. Ed. by Kokinov B. Holyoak K Gentner D. Sofia: New Bulgarian University, 1998, p. 405.

[16] G.E. Hinton. "Mapping part-whole hierarchies into connectionist networks". In: *Artificial Intellilgence* 46.1-2 (1990), pp. 47–75.

[17] Smolensky P. "Tensor product variable binding and the representation of symbolic structures in connectionist networks". In: *Artificial Intellilgence* 46.1-2 (1990), pp. 159–216.

[18] T. A. Plate. “Holographic Reduced Representations: Convolution algebra for compositional distributed representations”. In: *Proc. 12th int’l joint conference on artificial intelligence (IJCAI)*. Ed. by Reiter R. Mylopoulos J. San Mateo, CA: Kaufmann, 1991, pp. 30–35.

[19] T. A. Plate. *Holographic Reduced Representation: Distributed representation of cognitive structure*. Stanford, CA: CSLI, 2003.

[20] R. W. Gayler. “Vector symbolic architectures are a viable alternative for Jackendoff’s challenges”. In: *Behavioral and Brain Sciences* 29 (2006), pp. 78–79.

18 August 2016