Computing with High-Dimensional Vectors

Pentti Kanerva
University of California at Berkeley

Editor's note:
The author reviews the principles of high-dimensional (HD) computing as a brain-inspired paradigm, with variables and operations encoded in vectors with high dimensionality (e.g., 10,000). HD computing has been shown to be a robust novel approach with promising applications in language and biosignal processing.

—An Chen, Semiconductor Research Corporation

Computing at large

We usually think of computing as operating on bits, numbers, and memory pointers, and we build computers with circuits specialized for Boolean logic, arithmetic, and memory. Everything else is designed based on these basic operations by organizing them into programs. The genius of the von Neumann architecture is that the programs reside in memory and are manipulated with the same operations as the data—to the machine, the programs are data. The resulting flexibility has led to compiling and symbolic programming, which in turn has made it feasible to treat an endless assortment of tasks as computational.

The first digital computers were designed for calculation. They replaced rooms full of professionals called *computers* who worked with calculating machines and made use of logarithmic tables and such in their work. Digital computers soon eliminated the need for logarithmic tables and, over time, were used more and more for record-keeping, as we witnessed the era of mainframes. Both calculation and record-keeping require accuracy and reliability, and so computers are designed to work as deterministic machines: identical inputs always produce identical outputs. Although determinism is an unattainable ideal, the near-perfect operation can be achieved by the use of digital logic and wide design margins. Traditional theories of computing assume determinism and deal merely with inaccuracies arising from the precision with which numbers are represented, which gave birth to numerical analysis.

However, reliability at high speeds comes at a cost: it requires large amounts of energy. Decreasing the size of circuit elements has decreased the energy needed to operate them, but we are approaching a limit beyond which reduction in size makes the circuits unreliable. We can push physics only so far before things begin to break down. At the same time, computers are used increasingly in applications that do not require ultimate speed, precision, and reliability, or where the needed reliability is achieved by means other than ultrareliable circuit elements. Prime examples of these kinds of functions come from neuroscience.

Animal brains are the closest things in nature to resemble computers. As products of millions of years of evolution, they probably are nearly optimal for the function they serve. There are many things that brains do with apparent ease that we try to do with computers but only succeed partially at best. They concern behavior that allows animals

Digital Object Identifier 10.1109/MDAT.2018.2890221
Date of publication: 28 December 2018; date of current version: 29 May 2019.

to survive and prosper in an ever-changing world. In very general terms, our brains make it possible for us to understand and react to what is happening around us. What would this mean in computer terms? What conclusions can be drawn in regard to building computers for these and similar functions?

Cursory examination of the brain shows that its circuits are very large (fan-ins and fan-outs can be in the tens of thousands), the layout is not prescribed to the last detail, activity is widely distributed within a circuit and among different circuits, individual neurons need not be particularly reliable (neurons can die), and they operate with very little energy—the human brain runs on about 20 W. All these are so unlike how computers are built and operate that we need to rethink and extend our models of computing.

Models of computing inspired by the brain's circuits are as old as digital computers [1]. These models are called artificial neural nets, they have had a major role in cognitive science research, and they are now commercialized as deep learning nets. Neural net models are based on linear algebra and they compute with vectors and matrices. They are trained on standard computers with exacting algorithms such as gradient descent and principal components. Training is compute-intensive, although the final network can be fast and operate with very little energy. The networks are problem-specific and lack the flexibility and adaptivity that we associate with behaviors controlled by brains.

In this article, we consider the possibility of computing with high-dimensional random vectors as basic objects, contrasting it with conventional computing with bits and numbers. Computing of this kind was proposed by Plate in the early 1990s, as discussed thoroughly in the book *Holographic Reduced Representation: Distributed Representation of Cognitive Structure* [2] and condensed in a paper on "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors" [3].

The operations refer to vectors even as the vectors are realized with bits and numbers. There is an *addition* operation and a *multiplication* operation that preserve vector dimensionality. Furthermore, vectors make it possible to use the *permutation of coordinates* as a third operation—in fact, permutations give us a huge number of possible operations beyond addition and multiplication. The power of computing with high-dimensional vectors follows partly from high dimensionality (e.g., n = 10,000) and partly from the operations and their interaction with each other, i.e., from their algebra. The underlying math has much in common with linear algebra, but the approach is sufficiently different to deserve closer examination.

High-dimensional computing: An example

We will demonstrate the three operations with an example: identifying languages from their letter-use statistics (not relying on dictionaries). For each language, we compute a high-dimensional *profile vector* from about a million bytes of text. We use the same algorithm to compute a profile for a test sentence, which is then compared to the language profiles and the most similar one is chosen as the system's answer.

The profiles of our example are based on three-letter sequences called *trigrams* and they are computed as follows. First, the letters of the alphabets are assigned n-dimensional random vectors of +1s and −1s. The same *letter vectors* are used with all languages and test sentences. The letter vectors are used to make *trigram vectors* with permutation and multiplication. For example, the vector for the trigram *the* is computed by permuting the *t*-vector twice, permuting the *h*-vector once, taking the *e*-vector as is, and multiplying the three componentwise. This produces an n-dimensional trigram vector of randomly placed ±1s. Finally, the trigram vectors are added together to a *profile vector* by stepping through the text one trigram at a time. The result is an n-dimensional vector of integers. The cosine of such vectors can be used to measure their similarity.

Such an experiment with 21 European Union languages gave the following results [4]. All vectors were 10,000-dimensional. The language profiles clustered according to language families: Baltic, Germanic, Romance, and Slavic. When test-sentence profiles were compared to the language profiles, the correct language was chosen 97% of the time, and when the language profile for English was queried for the letter, most often following *th*, the answer was e. The mathematics underlying the algorithm is explained below.

High-dimensional vectors and robustness

We are used to the idea that if A is close to B and B is close to C, then A cannot be very far from C.

This intuition fails us totally in spaces with thousands of dimensions if we think in terms of the "territory" (number of vectors/points) within a given distance from point A. Doubling the distance in 2D space quadruples the territory, whereas in high-dimensional space it can increase billion-fold. For example, with 10,000-bit vectors, a mere billionth of those are within 4700 bits of any vector A, but nearly all are within 5300 bits. Increasing the Hamming distance from 4700 to 5300 increases the territory billion-fold. In practice, this means, for example, that if 10,000 bits are used to represent a person, over a third of the bits can change over time or be corrupted by noise or malfunction, and the resulting bit vector is still closer to the original than to a 10,000-bit representation of anyone else. This is true of high-dimensional spaces at large and is called *concentration of measure*.

Robustness of this kind is essential for animals in the natural world where things never repeat exactly and so it is essential for the brain to be able to identify similar sensory stimuli as the same (and dissimilar as different). The brain does so even when individual neurons function erratically or die. This ability can be based on representing the world and things in it with high-dimensional vectors and computing with such vectors. Once understood in principle, we can begin to build systems with similar capabilities.

To benefit from the robustness inherent in high dimensionality, a system must be able to identify new inputs with things already stored in the system's memory. The function is referred to as *associative memory*, which, in essence, is the nearest-neighbor search among vectors stored in the memory. Of engineering challenges presented by high-dimensional computing, the associative memory may prove out to be the most demanding.

Vector operations and their algebra

The first thing to stand out is that the vectors are not subdivided into nonoverlapping fields. Anything that is encoded into a vector is distributed equally over all its components. Thus, any part of a vector represents the same thing as the entire vector, only less reliably. Such a representation is called *holographic* or *holistic*. Yet, it is possible to encode and decode all data structures familiar from the ordinary computing—sets, sequences, and lists—with three operations on holographic vectors, namely, with addition, multiplication, and permutation, and with a measure of similarity. What exactly those operations are, depends on the nature of the vectors, whether binary, "bipolar," integer, real, or complex. In the example on language, we have used n-dimensional random bipolar vectors with equally probable +1s and –1s (vector mean equals 0): $A, B, C, \ldots \in \{1, -1\}^n$ ($n = 10{,}000$). The properties are shared by high-dimensional vectors of different kinds but are the simplest to demonstrate with the bipolar.

Similarity of vectors is based on distance. Hamming distance was used above to illustrate robustness. Dot product $A \cdot B$ can be used with bipolar vectors and it varies from $-n$ (the vectors are opposite) to n (they are the same). Cosine and (Pearson's) correlation can also be used, where cosine (or correlation) = 1 means that the vectors are maximally similar (the same), 0 means that they are maximally dissimilar or orthogonal, and –1 means that they are opposite. The cosine of two *randomly* chosen bipolar vectors is close to 0—the vectors are *quasiorthogonal*.

Addition (+) of bipolar vectors produces a vector of integers. The sum vector can be used as such in further computation, or returned to bipolar based on the sign and randomly turning 0s into ±1s. Some information is invariably lost when reverting to bipolar. The sum vector is *similar* to the argument vectors and independent of their order: $A + B + C \sim A, B, C$. It can therefore be used to represent a set or a multiset.

Multiplication (*) is done componentwise and it therefore commutes: $A * B = B * A$. The result is bipolar and dissimilar (quasiorthogonal) to the argument vectors: $A * B \nsim A, B$. However, multiplication preserves similarity: $(X * A) \cdot (X * B) = A \cdot B$ because this is true at every coordinate. A bipolar vector multiplied by itself produces a vector of 1s and so it is its own inverse. Multiplication distributes over addition: $X * (A + B + C + \ldots) = (X * A) + (X * B) + (X * C) + \ldots$ because it does so at every coordinate. However, if the sum vector is first turned to bipolar, distributivity holds only approximately.

The properties of multiplication make it useful for variable binding. For example, if X represents a variable and A its value, we can bind the two with $X * A$. Then, multiplication by X recovers the value: $X * (X * A) = (X * X) * A = A$, because X is its own inverse. We can also encode several variables and their values and superpose them into a single vector, as in $H = (X * A) + (Y * B) + (Z * C)$, such that

multiplying H by X recovers A, although only approximately, and so we must search the associative memory for A.

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

The second step in the equation follows from $*$ distributing over $+$. "Noise" denotes that the vectors are dissimilar to the ones that the system has in its memory.

Permutation rearranges vector coordinates, and a random permutation ρ produces a vector that is dissimilar to the argument vector: $\rho(A) \nsim A$. Like multiplication, permutation is invertible, $\rho^{-1}(\rho(A)) = A$, and it distributes over addition $\rho(A + B) = \rho(A) + \rho(B)$; it distributes also over multiplication $\rho(A * B) = \rho(A) * \rho(B)$ because addition and multiplication happen componentwise.

Permutations provide a means to represent sequences and nested structure. For example, the sequence (a, b, c) can be encoded as a sum

$$\begin{aligned} S_3 &= \rho(\rho(A)) + \rho(B) + C \\ &= \rho^2(A) + \rho(B) + C \end{aligned}$$

or as a product

$$P_3 = \rho^2(A) * \rho(B) * C$$

and extended to include d by first permuting S_3 or P_3: $S_4 = \rho(S_3) + D$ and $P_4 = \rho(P_3) * D$. The inverse permutation ρ^{-1} can then be used to find out, for example, the second vector in S_3

$$\begin{aligned} \rho^{-1}(S_3) &= \rho^{-1}(\rho^2(A)) + \rho^{-1}(\rho(B)) + \rho^{-1}(C) \\ &= \rho(A) + B + \rho^{-1}(C) \\ &= \text{noise} + B + \text{noise} \\ &\sim B \end{aligned}$$

or what comes after A and before C in P_3 by first canceling them out

$$\begin{aligned} &\rho^{-1}(P_3 * (\rho^2(A) * C)) \\ &= \rho^{-1}((\rho^2(A) * \rho(B) * (C) * (\rho^2(A) * C)) \\ &= \rho^{-1}(\rho(B) * (\rho^2(A) * \rho^2(A)) * (C * C)) \\ &= \rho^{-1}(\rho(B)) \\ &= B. \end{aligned}$$

If the pair (a, b) is encoded with two unrelated permutations ρ_1 and ρ_2 as $\rho_1(A) + \rho_2(B)$, then the nested structure $((a, b), (c, d))$ can be represented by

$$\begin{aligned} &\rho_1(\rho_1(A) + \rho_2(B)) + \rho_2(\rho_1(C) + \rho_2(D)) \\ &= \rho_{11}(A) + \rho_{12}(B) + \rho_{21}(C) + \rho_{22}(D), \end{aligned}$$

where ρ_{ij} is the permutation $\rho_i\rho_j$.

The versatility of computing with numbers is partly due to the fact that addition and multiplication form an algebraic structure called a field. We can expect computing with high-dimensional vectors to be equally powerful because addition and multiplication approximate a field and are complemented by permutations that combine with addition and multiplication in useful ways.

Second look at language identification: Working out the math

We introduced high-dimensional computing with an example where languages were identified from their trigrams statistics. Let us look at the algorithm in terms of the operations and their properties. The letters of the alphabet are represented by 10,000-dimensional random, independent, identically distributed, equally probable ±1 vectors A, B, C, . . ., which, therefore, are approximately orthogonal to each other. The trigram vectors that are made with permutation and multiplication are also approximately orthogonal to each other and to the letter vectors because both multiplication and permutation (rotation of coordinates was used in the example) produce vectors that are dissimilar to their arguments. However, addition produces vectors that are similar to their arguments, and therefore, a language profile vector resembles each of the trigram vectors added into it. That is why, similar histograms of trigrams produce similar language profiles—a language profile is nothing other than a histogram randomly projected to 10,000 dimensions. Finally, the letter most often following th in English is found by multiplying the profile for English with (the inverse of) $\rho^2(T) * \rho(H)$. The multiplication distributes over every trigram added into the profile and cancels out the initial th wherever it occurs. In particular, it releases E from the trigram vector for *the*, namely, from $\rho^2(T) * \rho(H) * E$; it also releases every other letter that comes after th, but since e is the most frequent, E has the highest dot product with the transformed profile. Its expected value is 10,000× the number of *e*s after th.

Note that the dot product is the same as between the profile vector and the trigram vector for *the*.

The example demonstrates the important aspects of high-dimensional computing. The same algorithm is used for training and for making profiles of test sentences. The algorithm is very simple, it is easily adapted to classification problems at large, and it works in a single pass over the data—the algorithm is incremental. Frequencies and probabilities can be recovered approximately from a profile vector by inverting the operations used to encode the profile—the representation is transparent.

Vectors other than bipolar

Demonstrating the operations with bipolar vectors is particularly simple because componentwise addition and multiplication use ordinary arithmetic. However, the idea is more general: the vectors can be binary or real or complex so long as they are high dimensional. Appropriate addition and multiplication operators exist for each kind, and all vectors can be permuted.

Systems based on binary vectors are the simplest to engineer. Dense binary (equally probable 0s and 1s) is mathematically equivalent to the bipolar: bipolar 1 maps to binary 0, addition becomes componentwise thresholded (majority) sum with a policy for breaking ties, multiplication becomes XOR, and similarity of vectors is based on the Hamming distance. Systems based on real vectors use components drawn randomly from a normal distribution with 0 mean and 1/n variance. Addition is an ordinary vector addition followed by normalization, multiplication is by circular convolution, and similarity of vectors is based on Euclidean distance between the normalized vectors. Systems based on complex vectors use random phase angles for components, addition is the vector addition followed by normalization, multiplication is componentwise addition of phase angles, and the similarity of two vectors is based on the length of their difference. High-dimensional computing with complex vectors is a likely model for computing with the timing of spikes relative to an underlying oscillation or clock.

These different frameworks are, in fact, related to each other. The equivalence of the binary to the bipolar has already been discussed. The real and the complex are related via Fast Fourier Transform, and the bipolar is equivalent to the complex when the phase angles are restricted to 0 and n [2].

Small-scale experiments, large-scale applications

High-dimensional computing has been demonstrated in experiments on language identification [4], [5] and biosignal classification [6], and is used commercially for making semantic vectors. The experiments have focused on the simplicity, transparency, and generality of the algorithms, their tolerance for component variability and failure, suitability for parallel execution, and energy efficiency.

Simplicity and transparency follow from the algebra of the operations, as pointed out in the previous section. We can argue for generality by observing that the operations, together with an associative memory, are sufficient for realizing a fully general programming language such as Lisp.

Robustness of high-dimensional computing is a consequence of extreme redundancy arising from distributed (holographic) representation where each coordinate computes the same thing independently of the others, although at very low precision. For example, ignoring half the coordinates allows the algorithms to work while producing less reliable results. Furthermore, independence of coordinates allows them to compute in parallel: in 10,000-dimensional, all 10,000 can add or multiply at once since there is no carry to be concerned with, as there is in arithmetic with number. Permutation is the only operation that operates across coordinates, by merely reordering them.

We have noted above the brain's unparalleled ability to learn and to adapt in a constantly changing world, as if it had the power of a supercomputer running on a tiny fraction of the energy. Brainlike computing does not require ultrareliable components switching at gigahertz rates. We can compute with very little energy if the rates are low and we can get by with components that vary and occasionally fail. A theory for such kind of computing will most likely have high-dimensional vectors at its core [7].

High-dimensional computing has been used commercially by a text analytics company Gavagai AB in Sweden, founded in 2008 (http://gavagai.se/about/). Its services are based on a high-dimensional vector algorithm called Random Indexing [8], which is a form of random projection. The algorithm embodies Distributional Hypothesis of linguistics in 2000-dimensional semantic vectors. The idea behind semantic vectors is that words with similar meaning have similar semantic vectors, useful in search engines,

for example. The algorithm works by assigning a random index vector—a 2000-dimensional random label—for each word in the vocabulary, analogous to the random letter vectors in the language-identification experiment. The random labels for words are then used to make semantic vectors for words by reading through the text. Each time a word appears, its semantic vector is updated by adding to it the random labels of adjacent words, usually two or three words on either side, referred to as a context window. As sum vectors, the semantic vectors are similar to the random labels of their context words, and thus, shared contexts produce similar semantic vectors.

Better-known semantic-vector algorithms include Latent Semantic Analysis (LSA) [9] and Word2vec [10], neither of which embraces high dimensionality and randomness. LSA makes several-hundred-dimensional semantic vectors with singular-value decomposition and becomes impractical when the vocabulary grows to 100,000s and the number of documents to millions. Word2vec deals better with large the data sets. Its semantic vectors are a solution to an optimization problem, formulated as the ability of words and phrases to predict the nearby words and phrases in large corpora of text. The algorithm involves explicit tabulation of frequencies and probabilities, and the problem is solved with the gradient descent in a high-dimensional vector space—the dimensionality is typically 1000 or less. In contrast, in computing with high-dimensional vectors based on their algebra, we try to avoid explicit tabulation of probabilities and calculation of gradients because of their computational cost, and also because it would introduce a batch process into the data flow.

All semantic vectors in use today are quite crude in fact. They capture elements of word meaning but are devoid of language structure. Structural analysis of language and the import of grammar to meaning have traditionally been the domain of symbolic AI. However, we can expect this to change, thanks to the ability of high-dimensional vector operations to encode (grammatical) structure. This is an obvious topic for research in language-understanding.

Prospects and opportunities

We began this article by highlighting the brain's ability to produce interesting behavior, vital to our survival, with slow, low-precision, unreliable components, and contrasted it with what we usually associate with computing power, namely, speed, precision, and reliability. Large memory capacity is one thing that brains and computers share and derive power from. If a synapse were to stand for 1 bit of information, the human brain would be the equivalent of 30 TB.

Whether or not computing with high-dimensional vectors explains the brain's computing, a host of new machine-learning algorithms can be founded on a computational algebra of the vectors. The algorithms can combine statistical learning from the data, with building and manipulating of the data structures. The algebra makes the algorithms transparent, thanks to the properties of the operations such as distributivity and invertibility. Transparency is often cited as a strength of rule-based systems and contrasted with the opacity of neural nets that are likened to black boxes.

High-dimensional vector operations are simple and lend themselves to incremental (online, "one-shot") learning, as seen in computing language profiles and semantic vectors. As already pointed out, they also lend themselves to a high degree of parallelism.

We have justified high-dimensional computing with examples from language and biosignal classification. These are obvious application areas, but the theory suggests more. When a signal from one source is represented in pseudorandom vectors in a high-dimensional space, it is easily combined with signals from other sources that are similarly represented, and their joint statistics can then be captured in high-dimensional vectors. Moreover, by being combined with invertible vector operations, contributions from different sources can be traced back to their origins. This suggests that high-dimensional computing can be used for multisensor integration, leading to applications in sensing, monitoring, control, and robotics.

The overall architecture would mimic neurobiology. It would consist of sensor-specific front-end processors, a high-dimensional vector processor, a high-dimensional memory, and effector-specific backend processors. The front-end processing would convert the raw sensory input into pseudorandom vectors, and backend processing would produce signals for driving motors, while integration and online learning would happen in the high-dimensional space using the vector operations.

Traditional neural nets and deep learning also have a place in this architecture, namely, as train-

able front-end and back-end processors. Since the signals for different sensory modalities—e.g., sight, sound, and touch—are so very different, they need very specific preprocessing to convert them into vectors suitable for integration. Training of such peripheral processors can be slow as long as the resulting system is fast. This again agrees with biology, when as children we learn the sounds of a language and as adults are unable to hear distinctions that another language makes and depends on. The strategy of combining deep learning with high-dimensional computing has been shown to work in preliminary experiments on image analysis.

The remarkable agreement between the requirements of high-dimensional computing and the nature of nanotechnology should not be overlooked. As electronic components become smaller, are built into ever larger circuits, and need to operate with little energy, their operation becomes prone to error. When used as components in traditional digital computers, every bit must be made ultrareliable. This is achieved by making the circuits redundant, but that costs in material and energy. There apparently is a physical limit beyond which further miniaturization no longer pays off.

Holographic representation is also redundant, but high reliability at bit level is unnecessary. That in itself leads to energy savings, which can be made even greater if we take advantage of analog properties of materials, such as programmable resistors. Operating with high-dimensional vectors can be benefitted from the storage elements with a wider range than binary.

Digital arithmetic and logic of conventional computers require complex circuits, which are built into an ALU that is physically separated from the memory. This creates the perpetual need to move the data between the two. Compared to digital arithmetic, the high-dimensional vector operations are very simple, and things like componentwise XOR can be built into the memory, eliminating the need to move 10,000-bit vectors into and out of an ALU. By distributing the arithmetic, this kind of "in-memory computing" saves both time and energy [6].

Conventional computing serves many areas exceedingly well. Then, there are areas where our long-standing efforts have not produced the results we had hoped for. In particular, mastering of tasks that rely on learning in everyday situations has proven particularly difficult. By computing with high-dimensional vectors, we hope to make inroads in such areas. To highlight the difference, the rocket that is sent to Jupiter's moon will be controlled by conventional computers, but the robot that explores the moon will have a brain that computes with high-dimensional vectors. ■

Acknowledgments

This work was supported in part by Japan's MITI Grant to Swedish Institute of Computer Science under Real World Computing (RWC) program; in part by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA; in part by Intel Strategic Research Alliance program on Neuromorphic Architectures for Mainstream Computing; and in part by NSF 16-526: Energy-Efficient Computing: from Devices to Architectures (E2CDA), a joint initiative between NSF and SRC.

■ References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[2] T. A. Plate, *Holographic Reduced Representation: Distributed Representation of Cognitive Structure*, Stanford, CA: CSLI Publications, 2003.

[3] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cogn. Comput.*, vol. 1, no. 2, pp. 139–159, 2009.

[4] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *Quantum Interaction, 10th International Conference, QI 2016*, J. A. de Barros, B. Coecke, and E. Pothos, Eds. Springer, 2017, pp. 265–274.

[5] M. Imani, J. Hwang, T. Rosing, A. Rahimi, and J. M. Rabaey, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Des. Test*, vol. 34, no. 6, pp. 94–101, Nov./Dec. 2017.

[6] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Biosignal processing with brain-inspired high-dimensional computing: Universal learning and classification for EMG and EEG," *in Proc. IEEE Spec. Iss. Non-Sil. Non-von Neum. Comp.*, in press.

[7] E. P. Frady, D. Kleyko, and F. T. Sommer, "A theory of sequence indexing and working memory in recurrent neural networks," *Neural Comput.*, vol. 30, no. 6, pp. 1449–1513, Jun. 2018.

[8] M. Sahlgren, A. Holst, and P. Kanerva, "Permutations as a means to encode order in word space," in *Proc. 30th Ann. Conf. Cogn. Sci. Soc.*, Austin, TX: Cognitive Science Society, 2008, pp. 1300–1305.

[9] T. K. Landauer and S. T. Dumais, "A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition induction, and representation of knowledge," *Psychol. Rev.*, vol. 104, no. 2, pp. 211–240, 1997.

[10] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neur. Info. Process. Syst.*, pp. 3111–3119, 2013.

Pentti Kanerva is a Research Scientist at Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA, USA. Kanerva has a PhD in philosophy from Stanford University, Stanford, CA, USA. He is a member of the Association for the Advancement of Artificial Intelligence.

■ Direct questions and comments about this article to Pentti Kanerva, Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA 94720-3198, USA; pkanerva@berkeley.edu.