

Integer Factorization with Compositional Distributed Representations

Denis Kleyko
UC Berkeley
Berkeley, CA, USA
Research Institutes of Sweden
Kista, Sweden

Connor Bybee
Christopher Kymn
Bruno Olshausen
UC Berkeley
Berkeley, CA, USA

Amir Khosrowshahi
Technology Development, Intel
Santa Clara, CA, USA

Dmitri E. Nikonov
Components Research, Intel
Hillsboro, OR, USA

Friedrich T. Sommer
E. Paxon Frady
Intel Labs
Santa Clara, CA, USA

ABSTRACT

In this paper, we present an approach to integer factorization using distributed representations formed with Vector Symbolic Architectures. The approach formulates integer factorization in a manner such that it can be solved using neural networks and potentially implemented on parallel neuromorphic hardware. We introduce a method for encoding numbers in distributed vector spaces and explain how the resonator network can solve the integer factorization problem. We evaluate the approach on factorization of semiprimes by measuring the factorization accuracy versus the scale of the problem. We also demonstrate how the proposed approach generalizes beyond the factorization of semiprimes; in principle, it can be used for factorization of any composite number. This work demonstrates how a well-known combinatorial search problem may be formulated and solved within the framework of Vector Symbolic Architectures, and it opens the door to solving similarly difficult problems in other domains.

KEYWORDS

Collective-State Computing, Hyperdimensional Computing, Vector Symbolic Architectures, Resonator network, Fractional Power Encoding, Integer Factorization

ACM Reference Format:

Denis Kleyko, Connor Bybee, Christopher Kymn, Bruno Olshausen, Amir Khosrowshahi, Dmitri E. Nikonov, Friedrich T. Sommer, and E. Paxon Frady. 2022. Integer Factorization with Compositional Distributed Representations. In *Neuro-Inspired Computational Elements Conference (NICE 2022)*, March 28-April 1, 2022, Virtual Event, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3517343.3517368>

1 INTRODUCTION

Distributed information processing and distributed representations were proposed in the 1980s for solving optimization and factorization problems. For example, associative networks [21] can solve optimization problems like the traveling salesman problem [22]. This research direction continues to draw interest for many reasons. First, it can provide insights into how biological neural circuits

solve optimization problems occurring in cognitive tasks. Second, distributed representations are well-suited for implementation in unconventional parallel hardware [24, 40], such as neuromorphic hardware [5, 14, 39], potentially providing scalable and low-energy solutions to challenging computing problems [6].

In this paper, we use a framework for forming structure-sensitive distributed representations that can flexibly encode compositional data structures [46] known as Vector Symbolic Architectures (VSA) a.k.a. Hyperdimensional Computing [13, 18, 25]. In VSAs, the essential operation for forming distributed representations of compositional data structures is the binding operation [29]. However, many VSA data structures require parsing, which amounts to a challenging combinatorial factorization problem that must be solved. Recently, resonator networks [10] were proposed that can efficiently factor compositional representations into their constituents [28]. Here, we demonstrate how the VSA technique *fractional power encoding* (FPE) [9, 11, 44] can be used to represent integers as vectors and we show how the problem of factorizing integers can be expressed as the problem of factorizing vectors. We then explain how resonator networks can be extended to solve prime factorization of integers, and we measure the performance and scaling of our method.

2 METHODS

2.1 Vector Symbolic Architectures and Fourier Holographic Reduced Representations

First in this section, we provide a brief overview of the required components from VSAs [29]. Please consult [32, 33] for a comprehensive survey. The key components of any VSA model are: a high-dimensional vector space where random vectors are pseudo-orthogonal (n denotes the dimensionality); symbol representations with randomized atomic vectors (a.k.a. hypervectors; bold lowercase letters, e.g., **a**); item memory for storing atomic hypervectors and performing auto-associative search (matrices indicated by bold uppercase letters, e.g., **A**).

Here, we are utilizing a version of VSA known as Fourier Holographic Reduced Representations (FHRR) [44]. In FHRR [44, 45], the atomic hypervectors are complex-valued random vectors, where each vector component can be considered as an angle (phasor) randomly and independently selected from the uniform distribution

over $(0, 2\pi]$ and with magnitude of one. The similarity measure is expressed by the normalized inner product between two phasor hypervectors (\mathbf{a} and \mathbf{b}) as $\frac{1}{n} \Re(\mathbf{a}^\dagger \mathbf{b})$, where \mathbf{a}^\dagger is the complex conjugate transpose, and \Re denotes the real part of the inner product. Each VSA model defines key operations used to manipulate atomic hypervectors. In FHRR, these are: *binding* (denoted as \odot), which is implemented as component-wise multiplication (Hadamard product); *inverse and unbinding*, which in FHRR corresponds to taking the complex conjugate of the vector to unbind (inverse $\bar{\mathbf{a}}/\mathbf{a}^{-1}$) and applying the Hadamard product ($\mathbf{b} \odot \bar{\mathbf{a}}$); *superposition*, (a.k.a. bundling, denoted as $+$) which is implemented as component-wise complex addition, possibly followed by some normalization function; and *permutation*, which can be implemented through a convolution operation or permutation (denoted as ρ but we do not use it here).

2.2 Fractional Power Encoding

In standard VSA methods, randomized atomic vectors act as symbols and can be manipulated like traditional symbolic representations. However, when sub-symbolic data or continuous values need to be represented (e.g., to solve machine learning problems [19, 30, 48]), it is important to form a similarity-preserving encoding. Fractional power encoding (FPE) is a method for such a similarity-preserving encoding, originally proposed in [44] (see Section 5.6) as a generalization of the fractional power vector [43]. This approach has recently received renewed interest for representing continuous manifolds, such as location in an environment [37], and has been connected to kernel methods for describing continuous functions [9, 11].

The idea behind the fractional power encoding starts with a single atomic hypervector \mathbf{z} . This vector can then be used to represent different integers through self-binding, where each self-binding step creates a new hypervector that is dissimilar to all the others. For instance, the value of 2 is represented by $\mathbf{z} \odot \mathbf{z}$; 3 is represented by $\mathbf{z} \odot \mathbf{z} \odot \mathbf{z}$, and so on. This can be expressed as exponentiating the vector (component-wise) with the integer value, i.e. the hypervector representation of integer i is formed as: $\mathbf{z}(i) = \mathbf{z}^i$.

It was recognized [44] that this exponentiation process can be defined continuously when using complex-valued FHRR vectors. Thus, the same scheme can be easily generalized to encoding any scalar x as: $\mathbf{z}(x) = \mathbf{z}^{\beta x}$, where we introduced a parameter β that controls the similarity-preserving properties of the resulting encoding. Hypervectors obtained with FPE preserve similarity between nearby values of x , while values further away have reduced similarity. The exact shape of this similarity metric defines the similarity kernel, and β regulates the width of this similarity kernel.

Another important property of FPE that we leverage is that it defines a systematic relationship between the binding operation and FPEs of scalars. In particular, binding the hypervectors representing two scalars x and y results in a hypervector representing $x + y$:

$$\mathbf{z}(x) \odot \mathbf{z}(y) = \mathbf{z}^{\beta x} \odot \mathbf{z}^{\beta y} = \mathbf{z}^{\beta(x+y)} = \mathbf{z}(x+y). \quad (1)$$

There is much more to be said about uses of FPE, including the representation of functions, the shape of the similarity kernel, and representations of multi-dimensional numerical data. We kindly refer interested readers to a recent thorough treatment of FPE in [9, 11].

2.3 Resonator Networks

In VSA, the representation of a conjunction of two or more hypervectors (e.g., \mathbf{a} , and \mathbf{b}) is achieved by binding: $\mathbf{s} = \mathbf{a} \odot \mathbf{b}$. The resulting hypervector \mathbf{s} is pseudo-orthogonal to the argument hypervectors (factors), and every combination of arguments results in a unique \mathbf{s} . The binding operation is invertible; when given all but one factor (\mathbf{b}), one can simply compute the unknown factor (\mathbf{a}) from the bound representation by unbinding: $\mathbf{s} \odot \bar{\mathbf{b}} = \mathbf{a} \odot \bar{\mathbf{b}} \odot \bar{\mathbf{b}} = \mathbf{a}$.

However, if none of the factors are given, then while decoding the vector is still feasible, it becomes a combinatorial search problem whose complexity grows exponentially with the number of factors. For instance, if there are 100 possibilities for factor \mathbf{a} and 100 for \mathbf{b} , then the challenge is to search over all 10,000 combinations of two factors. Recent work [10, 28] proposed an elegant mechanism, called the resonator network, to address the challenge of factorizing \mathbf{s} into its arguments.

To factor the representation from the input hypervector \mathbf{s} , the resonator network uses multiple populations, $\hat{\mathbf{a}}(t)$, and $\hat{\mathbf{b}}(t)$, each of which tries to infer a particular factor from the input hypervector. Each factor that is a possibility is stored in a separate factor item memory (\mathbf{A} , \mathbf{B}). Each population, called a resonator, communicates with the input hypervector and the other populations using the following dynamics:

$$\begin{aligned} \hat{\mathbf{a}}(t+1) &= f_n \left(\mathbf{A} \Re \left(\mathbf{A}^\dagger (\mathbf{s} \odot \bar{\hat{\mathbf{b}}}(t)) \right) \right); \\ \hat{\mathbf{b}}(t+1) &= f_n \left(\mathbf{B} \Re \left(\mathbf{B}^\dagger (\mathbf{s} \odot \bar{\hat{\mathbf{a}}}(t)) \right) \right). \end{aligned} \quad (2)$$

The process is iterative and progresses in discrete time steps, t . In essence at time t , each population can hold multiple weighted estimates for one of the factors through the VSA principle of superposition [12, 35]. This allows a population to test multiple guesses for factor identity simultaneously. Each resonator uses the current estimates from the other populations to invert the input hypervector and infer the factor of interest.¹ The cost of superposition is crosstalk noise, making the inference step noisy when many estimates are tested at once. Therefore, the next step uses the factor item memory to remove the extraneous estimates. The estimate for each factor is *cleaned up* by constraining the resonator activity only to the allowed atomic hypervectors stored in the corresponding factor item memory. Finally, a regularization step (denoted as $f_n(*)$) limiting the values of components of new estimates is needed. Successive iterations of this inference and clean up procedure (2) eliminate the noise as the factors become identified and find their place in the input vector. When the factors are fully identified, the resonator network reaches a stable equilibrium, and the factors can be deduced from the stable activity pattern.

3 RESULTS

3.1 Logarithmic FPE for encoding integers

We will use the problem of factorizing semiprimes (composite integers with exactly 2 prime factors) to demonstrate how FPE and resonator networks can interact together to solve this problem. A

¹Note that in (2) the update is synchronous, i.e., estimates at $t+1$ are based on the estimates from the previous t th iteration. It is possible to update estimates asynchronously. We use this asynchronous mode to perform the evaluation (see Section 3.1).

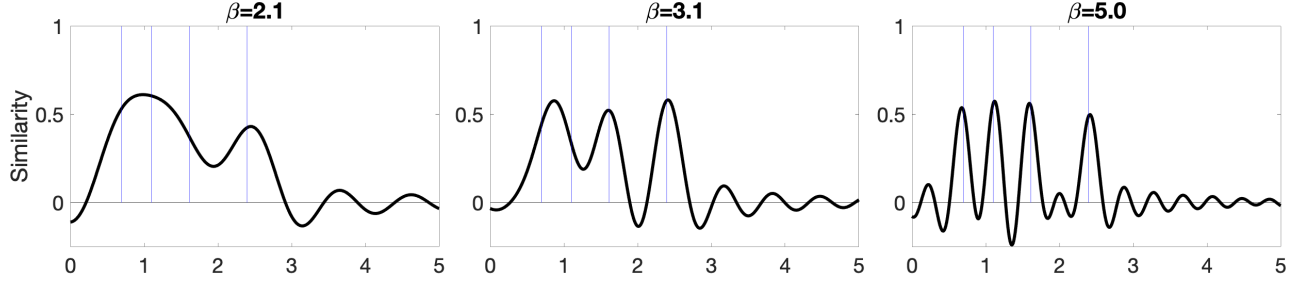


Figure 1: An example of varying behavior of the superposition of FPEs corresponding to a set of scalars: $\{\log(2), \log(3), \log(5), \log(11)\}$; y-axis depicts the average cosine similarity (thick lines) between the superposition hypervector and FPEs of scalars in x-axis. Thin vertical lines correspond to the locations of the elements of the considered set. The corresponding panels show cosine similarities for FPEs formed with different values of β . The reported values are averages computed from 30 simulation runs; $n = 512$.

semiprime s is obtained simply by multiplying two primes x and y :

$$s = xy. \quad (3)$$

We use $\mathcal{P}(s)$ to denote the set of all primes that are potential factors of s (since the minimum prime factor is two, all primes less than or equal to $s/2$ are possible factors). The i th element of the set is denoted as $\mathcal{P}(s)_i$. When setting up the factorization search, the set $\mathcal{P}(s)$ should be known. The set is used to form the item memory Φ containing hypervectors for all primes in $\mathcal{P}(s)$. The hypervector of $\mathcal{P}(s)_i$ (denoted as Φ_i) is formed with FPE (see Section 2.2) as:

$$\Phi_i = \mathbf{z}(\log(\mathcal{P}(s)_i)) = \mathbf{z}^{\beta \log(\mathcal{P}(s)_i)}. \quad (4)$$

Critically, the combination of the properties of the log transformation and FPE (see (1)) results in the following behavior of hypervectors:

$$\begin{aligned} \mathbf{z}(\log(s)) &= \mathbf{z}^{\beta \log(s)} = \mathbf{z}^{\beta \log(xy)} = \mathbf{z}^{\beta(\log(x) + \log(y))} \\ &= \mathbf{z}^{\beta \log(x)} \odot \mathbf{z}^{\beta \log(y)} = \mathbf{z}(\log(x)) \odot \mathbf{z}(\log(y)). \end{aligned} \quad (5)$$

In other words, the log transformation combined with FPE results in the binding between vectors being equivalent to the FPE vector of the product. This allows one to express the problem of semiprime factorization in terms of vector factorization, which can be solved through the resonator network formulation as in Section 2.3, where each resonator uses the codebook Φ .

3.2 FPEs in superposition

Traditionally in VSA, the most straightforward use of the superposition operation is to represent a set of elements [25, 34]. As with any other VSA representation, FPEs can also be used with the superposition operation to, e.g., represent a set of scalar values. However, due to the similarity-preserving properties of FPEs, the hypervector resulting from the superposition of several FPEs might exhibit counter-intuitive “hybrid” behavior, which is neither fully symbolic nor fully subsymbolic. When utilizing the resonator network to solve the factorization problem, we need to account for the behavior of FPE superpositions.

Consider an example of the following set:

$\{\log(2), \log(3), \log(5), \log(11)\}$, where the hypervector \mathbf{s} representing the set is formed using the FPEs corresponding to the elements

of the set as:

$$\mathbf{s} = \mathbf{z}^{\beta \log(2)} + \mathbf{z}^{\beta \log(3)} + \mathbf{z}^{\beta \log(5)} + \mathbf{z}^{\beta \log(11)}. \quad (6)$$

Fig. 1 presents the average cosine similarity between \mathbf{s} and the corresponding FPEs of scalars along the x-axis for different values of β . The first thing to notice in Fig. 1 is that the choice of β profoundly affects the obtained similarity distributions. When $\beta = 2.1$, FPEs of $\{\log(2), \log(3), \log(5)\}$ are similar to each other, and so when superimposed together, they interact in such a way that there is one large peak of similarity near their mean. Increasing β to 3.1 splits the large peak into two: one for $\log(5)$ and one in between $\log(2)$ and $\log(3)$. This is intuitive since $\log(2)$ and $\log(3)$ are closer to each other than $\log(3)$ and $\log(5)$. Finally, once β is large enough (e.g., $\beta = 5.0$) all four peaks become clearly distinct and they correspond to the values of the elements in the set.

Thus, setting the value of β allows traversing between two extremes: when β is very small, FPEs of scalars that are far away from each other are still very similar (subsymbolic behavior) while when β is very large, FPEs of scalars that are near each other are dissimilar (symbolic behavior). In other contexts, we expect that different applications might favor different modes. For example, when working with clustering problems [3, 20, 23, 31, 41], there is potential that subsymbolic merging would be useful for generating centroids and computing means. On the other hand, for integer factorization, it is important to choose β such that we only operate in the symbolic mode, as we desire that each different value is treated as a distinct alternative within the solution space of the factorization problem. Practically, this means we have chosen β to be large so that the inner product between the hypervectors of any two adjacent primes would be close to zero. This was achieved by setting β as :

$$\beta = \frac{10^4}{\min_i (\log(\mathcal{P}(s)_{i+1}) - \log(\mathcal{P}(s)_i))} \quad (7)$$

The extra factor of 10^4 was added to ensure that β was always sufficiently large.

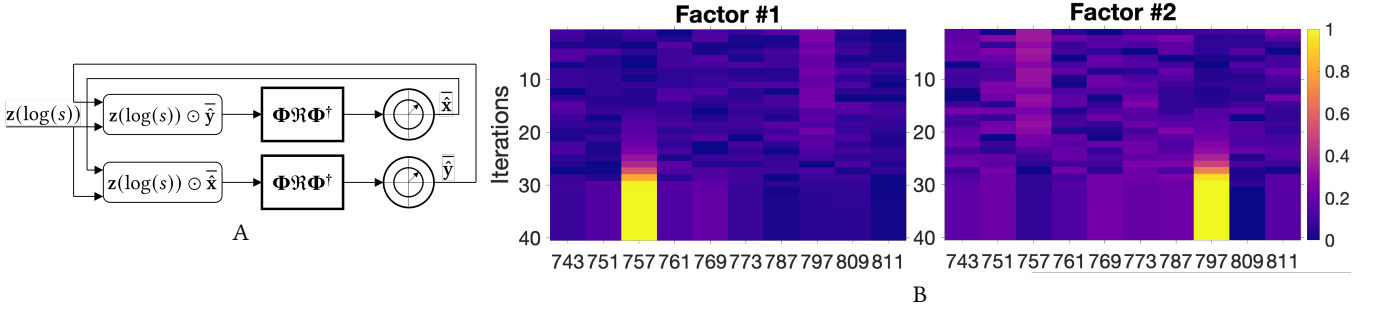


Figure 2: Left panel (A): an example of a resonator network with two factors for integer factorization according to (8). Each resonator uses the estimate from the other resonator to infer one of the factors (e.g. $z(\log(s)) \odot \hat{y}$), these estimates are then cleaned-up by limiting them to the span of the codebook ($\Phi \mathcal{R} \Phi^\dagger$), and finally the vector elements are restored to unit magnitude phasors ($f_n(x_i) = x_i/|x_i|$). Right panel (B): an example of convergence of a resonator network. Color values correspond to the normalized inner product between \hat{x} (Factor # 1) and \hat{y} (Factor # 2) and the entries of the codebook Φ corresponding to the primes depicted on x -axis. A small range of primes are shown for visualization. The dynamics are initially very chaotic until around iteration 30 where the network identifies the solution and quickly reaches a stable equilibrium.

3.3 Factorization of semiprimes with the resonator network

In this case, the dynamics of the resonator network factorizing s into x and y is described as follows:

$$\begin{aligned}\hat{x}(t+1) &= f_n\left(\Phi \mathcal{R}\left(\Phi^\dagger(z(\log(s)) \odot \hat{y}(t))\right)\right); \\ \hat{y}(t+1) &= f_n\left(\Phi \mathcal{R}\left(\Phi^\dagger(z(\log(s)) \odot \hat{x}(t+1))\right)\right),\end{aligned}\quad (8)$$

where $\hat{x}(t)$ and $\hat{y}(t)$ denote the hypervectors corresponding to the current estimates of the resonator network for $z(\log(x))$ and $z(\log(y))$; $f_n(x_i) = x_i/|x_i|$ normalizes each component to unit magnitude. Note that in (8) both factors use the same item memory Φ since both x and y are present in $\mathcal{P}(s)$. Once the resonator network converges or reaches the maximum number of iterations, the most recent estimates of the resonator network are used to obtain the predictions \hat{x} and \hat{y} corresponding to the primes in $\mathcal{P}(s)$ whose hypervectors in Φ are the most similar to $\hat{x}(t)$ and $\hat{y}(t)$. In the experiments, the maximum number of iterations was 100. A schematic overview of the resonator is shown in Fig. 2.

Let us walk-through an example of the factorization process described above. Assume that we would like to factorize semiprime $s = 603,329$ into its factors ($x = 757$ and $y = 797$). First, we need to define all the primes in $\mathcal{P}(s)$ that will be used to form the item memory Φ . In this case, the cardinality of $\mathcal{P}(s)$ will be $|\mathcal{P}(s)| = 26,135$ with the smallest prime being 2 and the largest one being 301,657. Once $\mathcal{P}(s)$ is fixed, we can use our equation for β (7) to calculate $\beta \approx 1.5 \times 10^9$. Next, we need to choose a suitable dimensionality of hypervectors n (see the next section for performance evaluation of different values of n). Then, a random n -dimensional base vector z is generated. The base vector z is used to populate the item memory Φ with hypervectors corresponding to FPEs of logarithms of elements of $\mathcal{P}(s)$ according to (4). We also form the FPE of the given semiprime s as $z(\log(s)) = z^{\beta \log(s)}$. The final step is to setup and run the resonator network according to (8) using the obtained $z(\log(s))$ and Φ . The initial estimates for $\hat{x}(0)$ and $\hat{y}(0)$ are set to the normalized superposition of all hypervectors in Φ . If n is large

enough, then after several iterations with high probability the resonator network will converge. The final state of $\hat{x}(t)$ and $\hat{y}(t)$ can be matched to the closest hypervectors in Φ , which will correspond to primes 757 and 797 (Fig. 2). Note that in this configuration, \hat{x} and \hat{y} can converge to either one of the primes.

3.4 Empirical evaluation of performance and scaling

In this section, we report the results of the empirical evaluation of the proposed approach. In the experiments below, we need to measure the success of the factorization by the average accuracy of correctly factorizing many semiprimes $s = xy$ (where x and y are chosen randomly from $\mathcal{P}(s)$).

First, we examine the factorization accuracy against the number of elements in $\mathcal{P}(s)$ and as a function of dimensionality, reported in left panel in Fig. 3. For each value of n , we can identify three regimes with respect to the accuracy: high-fidelity, where the accuracy is nearly perfect, low-fidelity, where the accuracy is not perfect but above chance, and random guessing, where the accuracy is effectively chance. It is evident that with increased n , the maximum size of $\mathcal{P}(s)$ within the high-fidelity regime also increased.

It is also important to estimate how the complexity of the approach scales. To do so, we consider the dimensionality of hypervectors required to perform the factorization successfully for the given cardinality of $\mathcal{P}(s)$. We have defined the successful factorization as the accuracy that is greater than or equal to 0.95. The scaling of the required dimensionality of hypervectors converges to a line with slope of approximately 1 with respect to cardinality of $\mathcal{P}(s)$ (central panel in Fig. 3). This observation is in line with the experiments reported in [28], where random hypervectors were used to form an abstract factorization problem. Practically, this also means that for large $|\mathcal{P}(s)|$, reasonable values of n would be sufficient to perform the factorization.

Recall that in Section 3.1, we discussed the choice of suitable β for a given $\mathcal{P}(s)$ to continue operating in the symbolic mode. Intuitively, the potential issue with scaling β is that when $\mathcal{P}(s)$

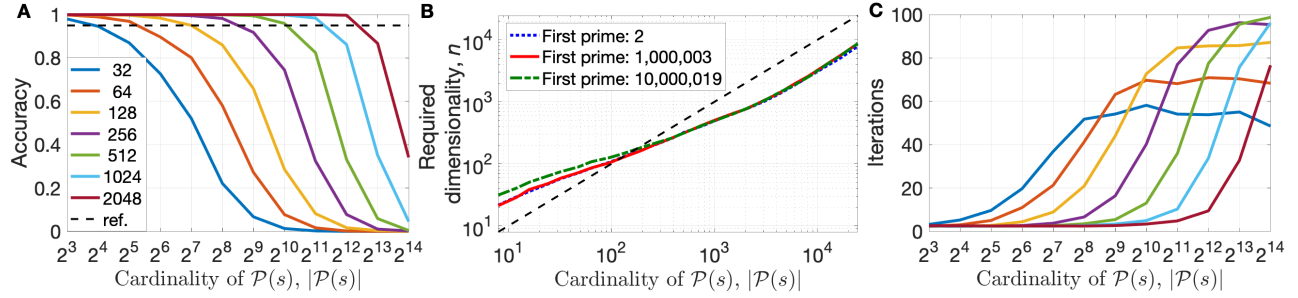


Figure 3: Left panel (A): the accuracy of semiprimes factorization against the cardinality of $\mathcal{P}(s)$. The reported values are averages computed from 4,000 random semiprimes. Central panel (B): average minimal dimensionality of hypervectors (y -axis) required to achieve at least 95% successful factorization for the given cardinality of $\mathcal{P}(s)$ (x -axis). Thin black dashed line depicts the linear relation between n and $|\mathcal{P}(s)|$. The colored lines correspond to different starting points used to form $\mathcal{P}(s)$. The reported values are averages computed from 10 simulation runs. During each simulation run, 1,000 randomly chosen semiprimes were used to assess the factorization accuracy for every considered value of n . Right panel (C): number of iterations used by the resonator network to either converge to a solution or to reach the maximum number of iterations (set to 100).

contains large primes, β will also be large due to the use of the log transformation. Large values of β could cause numerical issues when performing the FPE. In order to demonstrate the potential role of β on the factorization performance, Fig. 3B also depicts the required dimensionalities of hypervectors for primes in $\mathcal{P}(s)$ that were picked using different starting points, which following (7) leads to different values for β . First, the results suggest that the ranges of primes requiring the use of larger values of β did not incur a drastic increase in dimensionality of hypervectors, so factorization performance is mainly limited by the capacity of the resonator. Second, since larger values of β are not an issue, the proposed approach can handle varying ranges of primes.

In addition to the factorization accuracy, it is also worth looking at the average number of iterations used by the resonator network to converge (right panel in Fig. 3). There is a clear correspondence between the accuracy and the number of iterations of the resonator network. When the resonator network was in the high-fidelity regime, only a few iterations were required to find a solution, and $|\mathcal{P}(s)|$ increased the number of iterations also increased. Finally, once $|\mathcal{P}(s)|$ was too large for a chosen n , the resonator converged to the wrong answer or reached the iteration limit.

3.5 Factoring composite numbers beyond semiprimes

The proposed approach is not limited to semiprimes. In principle, it can be applied on any k -almost prime. In Fig. 4, we report the case of factorization of integers with three factors ($s = abc$) using a similar setup as above. Now there are three resonators, and each resonator is designed for three factors. They each have a similar update dynamics, for instance for the first factor:

$$\hat{\mathbf{a}}(t+1) = f_n \left(\Phi \mathbf{R} \left(\Phi^\dagger (\mathbf{z}(\log(s)) \odot \tilde{\mathbf{b}}(t)) \odot \tilde{\mathbf{c}}(t) \right) \right). \quad (9)$$

The observed results are consistent with that obtained for the case of semiprimes in Fig. 3. Note that compared to the case of semiprimes, for 3-almost primes larger values of n were required to get to the high-fidelity regime. This is expected since for the semiprimes the search space grows as $|\mathcal{P}(s)|^2$ while for 3-almost primes it grows

much faster as $|\mathcal{P}(s)|^3$. The approach can be extended to other composite numbers by including more resonators in the network. Again, the capacity and likelihood of solving the factorization problem depend on the combinatorics of the factors, and this grows exponentially with number of factors. Further, the identity vector (vector of all 1s, a.k.a. $\mathbf{z}(\log(1)) = \mathbf{z}^0$) can be added to Φ to enable solving problems with unknown number of factors.

4 DISCUSSION

4.1 Summary of the study

Our goal was to demonstrate that while Vector Symbolic Architectures (VSA) [25, 29] were originally proposed to solve problems in cognition, VSAs are a highly flexible and powerful framework for expressing challenging computational problems in high-dimensional vector spaces. We used integer factorization to showcase both the expressiveness of VSAs and novel techniques of representing numbers in high-dimensional vectors. VSAs are now well-known as frameworks for many novel computational devices that are designed for highly efficient and parallel computations [6, 47, 49]. Using VSAs to express challenging computational problems that can be solved by neural network architectures, like the resonator network, brings out the potential of utilizing neuromorphic hardware. It is relatively straight-forward to scale VSA algorithms like the resonator network – this simply means expanding the dimensionality of the vector representations. This ease of scalability is compatible with large scale meshes of neuromorphic chips [14]. While we did not execute these experiments on neuromorphic hardware, there are several previous models [7] and novel proposals [13, 15] that neuromorphically perform VSA computations.

One of our main contributions was expanding our understanding of how number systems can be expressed in vector spaces. Recently the technique of fractional power encoding (FPE) has been gaining new attention as a way to represent geometrical spaces, maps, manifolds and functions [9, 11, 38, 44]. Integers are the ordinal data type, and, therefore, their distributed representation should preserve the data topology, which provides a proper setup for the use of fractional power encoding. Previously integers were easily represented

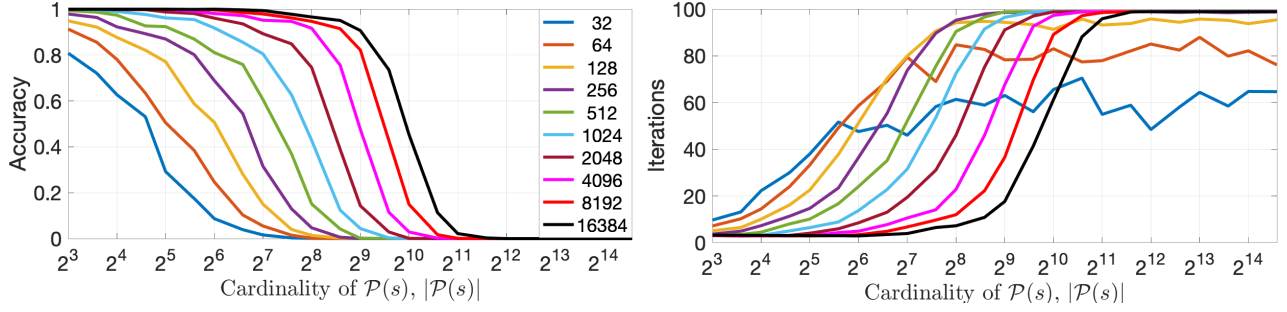


Figure 4: Left panel: the accuracy of 3-almost primes factorization against the cardinality of $\mathcal{P}(s)$. Right panel: number of iterations used by the resonator network to either converge to a solution or to reach the maximum number of iterations (set to 100). The values are averages from 1,000 randomly chosen 3-almost primes.

by the integer powers of an FPE, but in this formulation the binding operation results in the FPE of the integer sum. By expressing the FPE with the logarithms of integers, we enabled a representation of integer values where the binding operation now leads to the FPE of the integer product. With this formulation for representing integers, we could then express the integer factorization problem as the problem of vector factorization, which can be solved using resonator networks [10].

This FPE representation of logarithmic integers meant we needed to examine the consequences of superpositions of FPE vectors. The resonator network uses the principle of *search in superposition*, and for it to successfully solve the factorization problem, the individual factors need to be uniquely identifiable by their FPE vectors. We used the parameter β to rescale the FPE vectors so that the logarithmic FPE representations were sufficiently spaced such that their similarity kernels were not overlapping. There is a simple strategy for scaling β based on the minimum log distance between neighboring primes, but we also showed that there are few side-effects for making β much larger, and generally factorization performance is not too dependent on β as long as it is sufficiently large.

The way the factorization problem is solved by the resonator network is best described via the concept of *search in superposition*. During this process, many number combinations may be considered simultaneously, something that is not possible with conventional digital number representations. We believe that the extended idea of *computing in superposition* [29] is a particularly important aspect of VSAs that should be investigated further. It is also worth emphasizing, that the resonator network can be used beyond semiprimes (i.e., with more than two factors; see Section 3.4).

4.2 Related work

Our main contribution is a formalism for using VSAs to solve integer factorization, providing a way to solve classical factorization problems with distributed representations. Though our formalism benefits from the properties of VSAs (such as being distributed, robust, and computing in superposition), it was not our goal to demonstrate the superiority to other algorithms. The promise of our approach is really the potential of scalable and efficient execution of such an algorithm on neuromorphic hardware. It is important to keep in mind that the proposed approach should not be considered as a panacea in terms of providing a straight-forward polynomial

solution to the semiprimes factorization problem. The number of primes grows exponentially w.r.t. the number of bits used to represent a number, which also implies the exponential growth of the resonator.

This work expands on previous efforts to solve optimization and factorization problems with neural attractor networks and physics-based architectures. Early work proposed to design neural attractor networks based on matrix-type auto-associative memories [16, 17, 21, 36, 42] so that their dynamics is governed by a Lyapunov function that represents the objective of a particular optimization problem, for example, the path length in a traveling-salesman problem [22]. The fixed point attractor dynamics of such networks searches the solution space and settles at an approximate solution. The resonator network similarly settles at a solution if there is one. However, its dynamics is not governed by an energy landscape/Hamiltonian, if there is no solution, it can converge to limit cycles or chaotic orbits. It has been shown empirically that this richer dynamic repertoire accelerates the search and, as a result, outperforms gradient-based optimization significantly [28].

Quantum computing has been proposed as a physics-based method for solving combinatorial optimization problems [2, 50]. There are some similarities between quantum algorithms and the working principles of the resonator network used in this paper. The resonator network operates on sums of complex numbers to solve factorization problems, which can be regarded a classical analog of the superposition principle used in quantum computing. Adiabatic quantum annealing [2] methods solve an optimization problem by using the quantum tunneling effect. The optimization problem can be mapped to the Hamiltonian of a quantum system. The optimal solution or global minimum of the problem objective is found by slowly evolving the potential from an initial easy Hamiltonian to a more complicated Hamiltonian. Due to the challenge of building reliable and large quantum computers, there is a renewed interest in classical physics-based solvers, such as networks of coupled oscillators [1, 51]. The variables in the described resonator networks are complex-valued phasors, which can be represented by oscillators or by spiking neural networks [15].

4.3 Future work

There are several extensions of our approach, including subsequent analyses, that seem especially promising: While we presented the

algorithmic approach and its realization on the conventional parallel hardware (GPU), the real promise is in the implementing VSAs in neuromorphic hardware. To this date, it is still an open question how to implement a VSA system in such hardware in full. Probably the closest mapping to spiking hardware is provided via the Neural Engineering Framework [4, 8], although the potential challenge of this approach is spike efficiency. An alternative mapping proposal that is spike efficient is via representing FHRR phasors with spike times [15], but this approach is not yet extended to account for all VSA operations. Another promising hardware direction is in-memory computing [26, 27]. Since the above hardware is inherently noisy, it would provide a natural setup for demonstrating the robustness of our proposed factorization approach to noise (as expected from simulations performed in [28]). Another important direction is to design mapping of other difficult problems, such as the subset-sum problem and other combinatorial optimization problems, to resonator networks with FHRR. A particular challenge for designing mappings for other problems is the absence of strict guidelines directing mapping development, so for each problem the mapping has to be done ad hoc. Another limitation is that although resonator networks are well-suited for finding exact solutions (i.e., solving equality problems), it is less obvious how to formulate the problem of finding a maximum or minimum.

ACKNOWLEDGMENTS

FTS, BAO, CB, and DK were supported by Intel's THWAI. BAO and DK were supported by AFOSR FA9550-19-1-0241. DK was supported by the MSCA Fellowship (grant 839179). CJK was supported by the DoD through the NDSEG Fellowship. FTS was supported by Intel and NIH R01-EB026955.

REFERENCES

- [1] I. Ahmed, P.-W. Chiu, W. Moy, and C. H. Kim. 2021. A Probabilistic Compute Fabric Based on Coupled Ring Oscillators for Solving Combinatorial Optimization Problems. *IEEE Journal of Solid-State Circuits* (2021).
- [2] B. Apolloni et al. 1989. Quantum Stochastic Optimization. *Stochastic Processes and their Applications* 33, 2 (1989), 233–244.
- [3] T. Bandarakodla et al. 2019. Trajectory Clustering of Road Traffic in Urban Environments using Incremental Machine Learning in Combination with Hyperdimensional Computing. In *IEEE ITSC*. 1664–1670.
- [4] T. Bekolay, J. Bergstra, et al. 2014. Nengo: A Python Tool for Building Large-scale Functional Brain Models. *Frontiers in Neuroinformatics* 7 (2014), 1–13.
- [5] M. Davies, N. Srinivasa, T.-H. Lin, et al. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (2018), 82–99.
- [6] M. Davies, A. Wild, G. Orchard, et al. 2021. Advancing Neuromorphic Computing with Loihi: A Survey of Results and Outlook. *Proceedings of the IEEE* 109, 5 (2021), 911–934.
- [7] C. Eliasmith et al. 2012. A Large-scale Model of the Functioning Brain. *Science* 338, 6111 (2012), 1202–1205.
- [8] C. Eliasmith and C. H. Anderson. 2003. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. MIT Press.
- [9] E. P. Frady et al. 2021. Computing on Functions Using Randomized Vector Representations. *arXiv:2109.03429* (2021).
- [10] E. P. Frady, S. J. Kent, et al. 2020. Resonator Networks, 1: An Efficient Solution for Factoring High-Dimensional, Distributed Representations of Data Structures. *Neural Computation* 32, 12 (2020), 2311–2331.
- [11] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer. 2022. Computing on Functions Using Randomized Vector Representations (in brief). In *Neuro-Inspired Computational Elements Workshop (NICE)*. 1–12.
- [12] E. P. Frady, D. Kleyko, and F. T. Sommer. 2018. A Theory of Sequence Indexing and Working Memory in Recurrent Neural Networks. *Neural Computation* 30 (2018), 1449–1513.
- [13] E. P. Frady, D. Kleyko, and F. T. Sommer. 2021. Variable Binding for Sparse Distributed Representations: Theory and Applications. *IEEE Transactions on Neural Networks and Learning Systems* PP, 99 (2021), 1–14.
- [14] E. P. Frady, G. Orchard, et al. 2020. Neuromorphic Nearest-Neighbor Search Using Intel's Pohoiki Springs. In *Neuro-Inspired Computational Elements Workshop (NICE)*. 1–10.
- [15] E. P. Frady and F. T. Sommer. 2019. Robust Computation with Rhythmic Spike Patterns. *Proceedings of the National Academy of Sciences* 116, 36 (2019), 18050–18059.
- [16] A. A. Frolov et al. 2002. On Informational Characteristics of Willshaw-like Auto-associative Memory. *Neural Network World* 12, 2 (2002), 141–157.
- [17] A. A. Frolov et al. 2006. Time of Searching for Similar Binary Vectors in Associative Memory. *Cybernetics and Systems Analysis* 42, 5 (2006), 615–623.
- [18] R. W. Gayler. 2003. Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience. In *Joint International Conference on Cognitive Science (ICCS/ASCS)*. 133–138.
- [19] L. Ge and K. K. Parhi. 2020. Classification using Hyperdimensional Computing: A Review. *IEEE Circuits and Systems Magazine* 20, 2 (2020), 30–47.
- [20] A. Hernández-Cano, Y. Kim, and M. Imani. 2021. A Framework for Efficient and Binary Clustering in High-Dimensional Space. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. 1859–1864.
- [21] J. J. Hopfield. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences* 79, 8 (1982), 2554–2558.
- [22] J. J. Hopfield and D. W. Tank. 1985. “Neural” Computation of Decisions in Optimization Problems. *Biological Cybernetics* 52, 3 (1985), 141–152.
- [23] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing. 2019. HDCluster: An Accurate Clustering Using Brain-Inspired High-Dimensional Computing. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. 1591–1594.
- [24] H. Jaeger. 2021. Towards a Generalized Theory Comprising Digital, Neuromorphic and Unconventional Computing. *Neuromorphic Computing and Engineering* 1, 1 (2021), 1–38.
- [25] P. Kanerva. 2009. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation* 1, 2 (2009), 139–159.
- [26] G. Karunaratne et al. 2020. In-Memory Hyperdimensional Computing. *Nature Electronics* 3, 6 (2020), 327–337.
- [27] G. Karunaratne, M. Schmuck, M. Le Gallo, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi. 2021. Robust High-dimensional Memory-augmented Neural Networks. *Nature Communications* 12, 1 (2021), 1–12.
- [28] S. J. Kent, E. P. Frady, F. T. Sommer, and B. A. Olshausen. 2020. Resonator Networks, 2: Factorization Performance and Capacity Compared to Optimization-Based Methods. *Neural Computation* 32, 12 (2020), 2332–2388.
- [29] D. Kleyko, M. Davies, E. P. Frady, et al. 2021. Vector Symbolic Architectures as a Computing Framework for Nanoscale Hardware. *arXiv:2106.05268* (2021), 1–28.
- [30] D. Kleyko, M. Kheffache, et al. 2021. Density Encoding Enables Resource-Efficient Randomly Connected Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 8 (2021), 3777–3783.
- [31] D. Kleyko, E. Osipov, et al. 2019. Distributed Representation of n-gram Statistics for Boosting Self-Organizing Maps with Hyperdimensional Computing. In *International Conference on Perspectives of System Informatics (PSI)*. 64–79.
- [32] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi. 2021. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations. *arXiv:2111.06077* (2021), 1–27.
- [33] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi. 2021. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part II: Applications, Cognitive Models, and Challenges. *arXiv:2112.15424* (2021), 1–36.
- [34] D. Kleyko, A. Rahimi, R. W. Gayler, and E. Osipov. 2020. Autoscaling Bloom Filter: Controlling Trade-off Between True and False Positives. *Neural Computing and Applications* 32 (2020), 3675–3684.
- [35] D. Kleyko, A. Rosato, E. P. Frady, M. Panella, and F. T. Sommer. 2020. Perceptron Theory for Predicting the Accuracy of Neural Networks. *arXiv:2012.07881* (2020), 1–12.
- [36] A. Knoblauch et al. 2010. Memory Capacities for Synaptic and Structural Plasticity. *Neural Computation* 22, 2 (2010).
- [37] B. Komer and C. Eliasmith. 2020. Efficient Navigation using a Scalable, Biologically Inspired Spatial Representation. In *Annual Meeting of the Cognitive Science Society (CogSci)*. 1532–1538.
- [38] B. Komer, T. C. Stewart, A. R. Voelker, and C. Eliasmith. 2019. A Neural Representation of Continuous Space using Fractional Binding. In *Annual Meeting of the Cognitive Science Society (CogSci)*. 2038–2043.
- [39] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, et al. 2014. A Million Spiking-neuron Integrated Circuit with a Scalable Communication Network and Interface. *Science* 345, 6197 (2014), 668–673.
- [40] R. Moughan. 2021. *Parallel Architectures for Hyperdimensional Computing*. Master's thesis. UC Berkeley.
- [41] E. Osipov, S. Kahawala, et al. 2021. HyperSeed: Unsupervised Learning with Vector Symbolic Architectures. *arXiv:2110.08343* (2021), 1–12.
- [42] G. Palm. 1980. On Associative Memory. *Biological Cybernetics* 36, 1 (1980), 19–31.
- [43] T. A. Plate. 1992. Holographic Recurrent Networks. In *Neural Information Processing Systems*. 34–41.

- [44] T. A. Plate. 1994. *Distributed Representations and Nested Compositional Structure*. University of Toronto, PhD Thesis.
- [45] T. A. Plate. 1995. Holographic Reduced Representations. *IEEE Transactions on Neural Networks* 6, 3 (1995), 623–641.
- [46] D. A. Rachkovskij. 2001. Representation and Processing of Structures with Binary Sparse Distributed Codes. *IEEE Transactions on Knowledge and Data Engineering* 3, 2 (2001), 261–276.
- [47] A. Rahimi, S. Datta, et al. 2017. High-dimensional Computing as a Nanoscalable Paradigm. *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, 9 (2017), 2508–2521.
- [48] A. Rahimi, P. Kanerva, et al. 2019. Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals. *Proc. IEEE* 107, 1 (2019), 123–143.
- [49] A. Rahimi and B. Recht. 2007. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 20. 1–8.
- [50] P. W. Shor. 1999. Polynomial-time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review* 41, 2 (1999), 303–332.
- [51] T. Wang and J. Roychowdhury. 2019. OIM: Oscillator-based Ising Machines for Solving Combinatorial Optimisation Problems. In *International Conference on Unconventional Computation and Natural Computation (UCNC)*. 232–256.