No two brains are alike: Cloning a hyperdimensional associative memory using cellular automata computations

Denis Kleyko[1] *and Evgeny Osipov[1]

Luleå University of Technology, 971 87 Luleå, Sweden

Abstract
This paper looks beyond of the current focus of research on biologically inspired cognitive systems and considers the problem of replication of its learned functionality. The considered challenge is to replicate the learned knowledge such that uniqueness of the internal symbolic representations is guaranteed. This article takes a neurological argument "no two brains are alike" and suggests an architecture for mapping a content of the trained associative memory built using principles of hyperdimensional computing and Vector Symbolic Architectures into a new and orthogonal basis of atomic symbols. This is done with the help of computations on cellular automata. The results of this article open a way towards a secure usage of cognitive architectures in a variety of practical application domains.

Keywords: Vector Symbolic Architectures, distributed representation, knowledge transfer, hyperdimensional computing, cellular automata, cognitive systems

1 Introduction

Alternative computing architectures with autonomous continuously learning capabilities and an ability to generalize are highly demanded in many practical domains including robotics, intelligent industries, home automation, and e-health. Biologically Inspired Cognitive Architectures (BICA) by definition satisfy such requirements and as such have a high potential to emerge as a solution for many applications including the life- and mission critical ones.

While in the context of BICA the current research focus is on educational processes of cognitive systems or robots this article suggests to look beyond this phase and consider the knowledge replication and transfer functionality. It is exactly this functionality, which will make BICA-based technical systems practically usable in technical applications.

Consider a simple scenario where, for example, a BICA system is installed for home automation in a facility with a large number of initially identical rooms. Suppose that one BICA system is installed per room and its purpose is to learn the behavioral pattern of the users in the particular room and adjust the smart functionality accordingly. Obviously, it is unacceptable (or at least impractical) that each installed system is completely blank in the beginning

*Corresponding author. Email: `denis.kleyko@ltu.se; evgeny.osipov@ltu.se` and need to be trained from the scratch. Instead, it is safe to assume that the system is pre-trained on typical behavioral patterns, which then will be replicated in several copies, which will continue their independent functionality as the system continues to operate.

From the biological prospective, the challenge associated with the replication of the cognitive content is that while the two copies must be functionally same, they should be based on unique internal representations. Neurological evidence for this argument is presented in [1]. At the same time, similar behavior emerges from different initial states of neural tissue. Paraphrasing this argument: "No two brains are alike". Similar reasoning is supported by philosophical studies in [2], which in a simplified manner could be summarized as: The result of copying of a mind are essentially two independent minds living their own lives.

The argument for "uniqueness of the copies" has a meaning with respect to technical requirements from potential users of the BICA systems: the security of systems operation. In the presented above practical home automation use-case a simple copying makes the entire system extremely vulnerable to malicious attacks. Once the BICA-origin is compromised (e.g., the internal representations of atomic concepts become known), all subsequent copies are compromised as well. If not addressed, this will remain one of the major arguments against using BICA-based systems in the (life-critical) technical applications.

This article focuses on the challenge of replication of the content of the long-term memory of the BICA system. Specifically it is assumed that the long-term memory is implemented using the principles of hyperdimensional computing and Vector Symbolic Architectures (VSAs). VSAs [3] is a class of connectionist models for representing structured knowledge and implementing analogical-based reasoning. It is argued that VSAs could be a suitable candidate for implementing functionality of general artificial intelligence [3]. VSAs also constitute a significant part of the Semantic Pointer Architecture [4], which exhibits a wide range of cognitive functionality.

The core contribution of the article is the pipeline for cloning the content of the learned VSA-based BICA, which satisfy the "no two brains are alike" argument. Technically, the described solution allows to map the trained memory to a new unique basis while retaining the internal relationships between the symbols represented in hyperdimensional space. The mapping is done with the help of computations on cellular automata (CA). Rule 90 of CA is chosen as it possesses several properties highly relevant for achieving the objectives of the considered mapping task.

This paper is structured as follows. Section 2 introduces high-dimensional vectors and operations on them. Cellular automata are briefly described in Section 3. Section 4 presents the main contribution of this paper – a pipeline for mapping VSA-based memory into a new unique basis with cellular automata computations. The conclusions are presented in Section 5.

2 High-dimensional vectors and related operations

Hyperdimensional computing (also known as Vector Symbolic Architectures, VSAs) operates with high-dimensional vectors (also referred to as HD vectors or distributed representations). There are several different types of VSAs, each using different representations, e.g. [5, 6, 7, 8, 9]. This paper considers only Binary Spatter Code variant of VSAs [5], in which the individual elements only take the binary values "0" or "1". Information in VSAs is represented in a distributed fashion, where a single, unitary entitity is represented as an HD vector. The values of each element of an HD vector are independent of each other, and "0" and "1" values have approximately the same density, i.e. $p_1 \approx p_0 \approx 0.5$.

The similarity between two HD vectors is characterized by the Hamming distance. It measures the number of positions in the two compared HD vectors in which they have different values:

$$d_h(\mathbf{a}, \mathbf{b}) = \frac{\|\mathbf{a} \oplus \mathbf{b}\|_1}{N} = \frac{\sum_{i=0}^{N-1} a_i \oplus b_i}{N}, \tag{1}$$

where a_i, b_i are values of ith element of HD vectors $\mathbf{a}$ and $\mathbf{b}$ of dimension N, $\|.\|_1$ is the Hamming weight (the count of elements having the value 1), and $\oplus$ denotes the elementwise XOR operation.

In very high dimensional spaces, the Hamming distance between two arbitrarily chosen HD vectors is concentrated around 0.5. That is, arbitrarily chosen HD vectors are, with overwhelmingly high probability, almost orthogonal (i.e. effectively dissimilar). This is similar to the behaviour of symbolic representations – arbitrarily chosen symbols are generally different. Interested readers are referred to [5] and [10] for comprehensive analysis of probabilistic properties of the hyperdimensional representation space. There are three common operations with HD vectors: binding, bundling, and permutation.

Bundling of vectors. The bundling operation is used to join multiple entities into one structure; it is implemented by a majority rule of the HD vectors representing the entities. A elementwise thresholded sum of n vectors yields "0" when $n/2$ or more arguments are "0", and "1" otherwise. If the sum produces an even number, the resulting tie is broken randomly. This is equivalent to adding an extra random HD vectors [5]. The operation is denoted as the sum $[\mathbf{a} + \mathbf{b} + \mathbf{c}]$.

Binding of vectors. Binding, which can be interpreted as assigning a value to a field, is implemented as the elementwise XOR operation (denoted as $\oplus$) on the corresponding HD vectors.

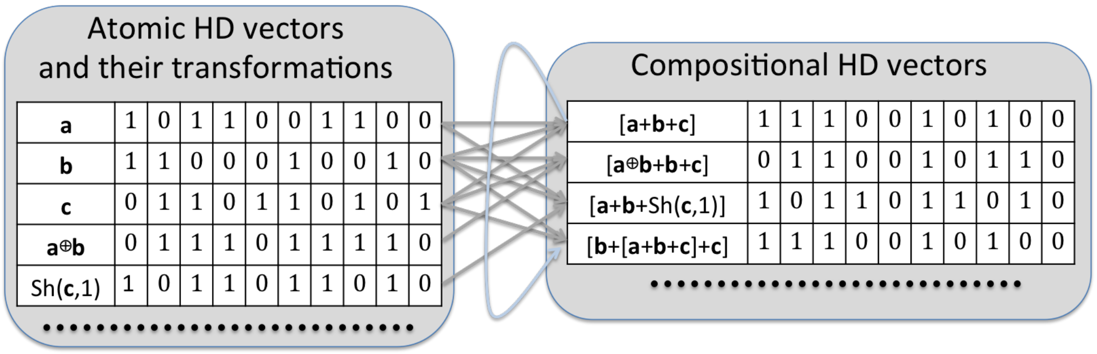
Permutation of vectors. Permutation produces an HD vector dissimilar to the permuted one (i.e. the normalized Hamming distance between them equals approximately 0.5). Several random HD vectors can be generated from a single vector through different permutations. The cyclic shift operation is a special case of the permutation [5]. The result of operation is obtained by cyclically shifting $\mathbf{a}$ by i elements and denoted here as $\mathrm{Sh}(A, i)$.

An example of operation which can be done with HD vectors is called holistic mapping. It can be used to answer non-trivial queries (e.g., "What is the dollar of Mexico?") by operating on the whole representation [11].

2.1 VSA-based BICA

Hyperdimensional computing and Vector Symbolic Architectures are used to represent structured knowledge and implement analogical reasoning. The VSAs based memory is normally structured in two parts: a clean-up memory (often referred to as item-memory) and a compositional memory. The structure is exemplified in Figure 1. The item memory normally contains atomic HD vectors and is normally used for detailed analysis of VSAs aggregates. For the purposes of future discussion we also insert results of binding and permutation operations generated during system's operation. Thus, the HD vectors stored in the item-memory serve as components for all other HD vectors in the system.

The second part of the memory is the compositional memory. In terms of symbolic systems compositional memory will contain meaningful transformations of the atomic symbols and their bindings. The symbols in this memory are formed by bundling several components from the item memory. VSA structures can be formed from atomic HD vectors only (e.g. $[\mathbf{a} + \mathbf{b} + \mathbf{c}]$), be a result of bundling of binding or permutation of atomic HD vectors (e.g. $[\mathbf{a} \oplus \mathbf{b} + \mathbf{b} + \mathbf{c}]$, $[\mathbf{a} + \mathbf{b} + \mathrm{Sh}(\mathbf{c}, 1)]$), and other compositional HD vectors as well (e.g. $[\mathbf{b} + [\mathbf{a} + \mathbf{b} + \mathbf{c}] + \mathbf{c}]$).



a	1	0	1	1	0	0	1	1	0	0
b	1	1	0	0	0	1	0	0	1	0
c	0	1	1	0	1	1	0	1	0	1
a⊕b	0	1	1	1	0	1	1	1	1	0
Sh(**c**,1)	1	0	1	1	0	1	1	0	1	0

[**a**+**b**+**c**]	1	1	1	0	0	1	0	1	0	0
[**a⊕b**+**b**+**c**]	0	1	1	0	0	1	0	1	1	0
[**a**+**b**+Sh(**c**,1)]	1	0	1	1	0	1	1	0	1	0
[**b**+[**a**+**b**+**c**]+**c**]	1	1	1	0	0	1	0	1	0	0

Figure 1: An example of an item memory with three atomic HD vectors (**a**, **b**, and **c**), a binding HD vector ($\mathbf{a} \oplus \mathbf{b}$) and a permutated vector $\mathrm{Sh}(\mathbf{c}, 1)$). The compositional memory (right-hand side) contains four compositional HD vectors. For simplicity of the presentation, the system is exemplified on 10-dimensional vectors. In the simulations HD vectors with 10,000 elements were used.

3 Cellular Automata

A cellular automaton (CA) is a discrete computational model consisting of a regular grid of cells. Each cell can be in one of a finite number of states (two - for the binary automaton). States of cells evolve in discrete time steps according to a fixed rule. The state of a cell on the next computational step depends on its current state and states of its neighbors. Notably, amongst the rules of CA there are those (e.g., rule 110), which make CA operate at the edge of chaos and were proven to be Turing complete. Another interesting rule which is employed in this paper is rule 90. For brevity of presentation the 90th rule of cellular automata will be denoted as CA_{90}. Also notation CA_{90}^{n} will be used to denote the n-th computing step of the cellular automaton. Albeit this rule is not Turing complete, it posses several properties relevant for the scope of this paper.

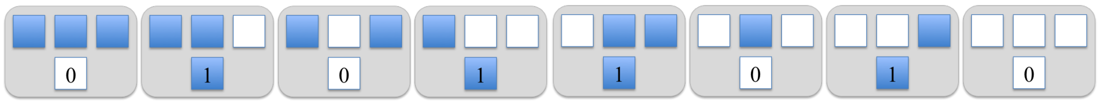


Figure 2: The assignment of new states for a center cell when the cellular automaton uses rule 90. A hollow cell corresponds to zero state while a shaded cell marks one state.

The computations performed by cellular automata are local. The new state of a cell is determined by previous states of the cell itself and its two neighboring cells (left and right). Thus only three cells are involved in a computation step, i.e. for binary states there are in total $2^3 = 8$ combinations. The rule assigns states for each of eight combinations. Figure 2 illustrates all combinations and corresponding states for CA_{90}. It can be seen from the figure that CA_{90} assigns the next state of a central cell solely on the previous states of the neighboring cells. In particular, the new state is the result of XOR operation on the states of the neighboring cells. Thus, CA_{90} has a computational advantage since CA implementation can be vectorized. That is if at time step t vector $\mathbf{x}(t)$ describes the states of CA, then the states for the next time step $t + 1$ are computed as follows:

$$\mathbf{x}(t+1) = \mathrm{Sh}(\mathbf{x}(t), 1) \oplus \mathrm{Sh}(\mathbf{x}(t), -1). \tag{2}$$

where Sh() is the notation for cyclic shift operation.

3.1 CA and hyperdimensional vectors

In the scope of VSAs and hyperdimensional computing CA (rules 90 and 110) were recently explored in [12, 13] for projecting binarized features into hyperdimensional space. Further, in [14] this approach was applied for modality classification of medical images. In the present work, we utilize three major properties of CA rule 90:

1. Random projection;
2. Preservation of the binding operation;
3. Preservation of the cyclic shift.

The first property means that when CA_{90} is initialized with a random HD vector HD_{INIT}, its evolved state is another HD vector of the same density ($p_1 \approx p_0 \approx 0.5$). Moreover this vector is dissimilar with the original initialization vector. That is $d_h(HD_{INIT}, CA^n_{90}(HD_{INIT})) \approx 0.5$.

The second property tells that if HD vector $\mathbf{c}$ is the result of binding of two other HD vectors: $\mathbf{c} = \mathbf{a} \oplus \mathbf{b}$ then after n computational steps of CA_{90}, the evolved HD vector $CA^n_{90}(c)$ is the result of binding of the evolved HD vectors $CA^n_{90}(a)$ and $CA^n_{90}(b)$ for the same number of steps. That is $d_h(CA^n_{90}(c), CA^n_{90}(a) \oplus CA^n_{90}(b)) = 0$.

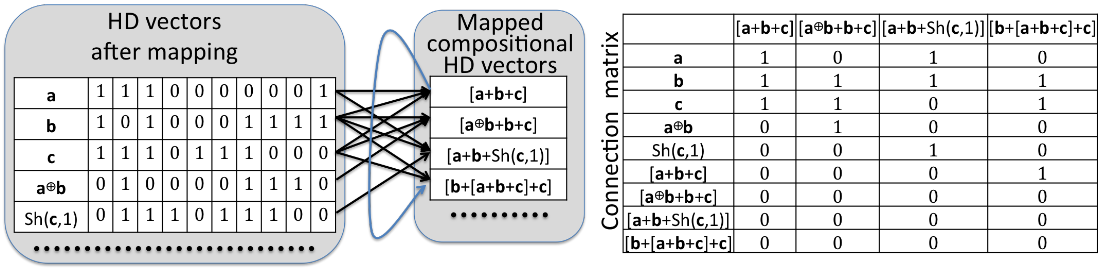
Finally, computations with CA_{90} preserve a special case of the permutation operation on HD vectors - cyclic shift by i elements. Suppose $\mathbf{d} = \mathrm{Sh}(\mathbf{a}, i)$. After n computational steps of CA_{90}, the cyclic shift of the evolved HD vector $CA^n_{90}(a)$ by i elements equals the evolved shifted HD vector $CA^n_{90}(d)$. That is $d_h(CA^n_{90}(d), Sh(CA^n_{90}(a, i)) = 0$.

These properties of CA_{90} constitute the core of the memory cloning pipeline presented in the next section.

4 Cloning of VSA-based associative memory with CA_{90} computations

The challenge with the replication of the VSA-based memory into a new unique basis comes from the fact that CA_{90} does not preserve the bundling operation. That is the result of the evolution of the bundled HD vector (computed as the majority sum of several HD vectors) becomes dissimilar to its CA-evolved components. That is why additional architectural solution is needed.

The core of the proposed memory cloning pipeline is the mechanism for *describing* relationships between HD vectors from the item-memory, which result in the items of the compositional memory. The most straightforward way to describe such relationships is to introduce a *relation matrix*. This matrix simply marks which items are included in the bundles. The idea is illustrated in Figure 3. In the left part of the figure the arrows show the contribution of each HD vector from the item-memory to the particular compositional HD vector. The table on the right is the relation matrix representing these relationships in the binary form. Note, that compositional structures are also valid symbols to be used in other VSA-compositions. This is depicted by the lighter arrow originating and ending up in the compositional VSA memory.



a	1	1	1	0	0	0	0	0	0	1
b	1	0	1	0	0	0	1	1	1	1
c	1	1	1	0	1	1	1	0	0	0
a⊕b	0	1	0	0	0	0	1	1	1	0
Sh(**c**,1)	0	1	1	1	0	1	1	1	0	0

	[a+b+c]	[a⊕b+b+c]	[a+b+Sh(c,1)]	[b+[a+b+c]+c]
a	1	0	1	0
b	1	1	1	1
c	1	1	0	1
a⊕b	0	1	0	0
Sh(**c**,1)	0	0	1	0
[**a**+**b**+**c**]	0	0	0	1
[**a**⊕**b**+**b**+**c**]	0	0	0	0
[**a**+**b**+Sh(**c**,1)]	0	0	0	0
[**b**+[**a**+**b**+**c**]+**c**]	0	0	0	0

Figure 3: Recomputing compositional HD vectors using connection matrices and mapped atomic HD vectors.

The role of the relation matrix is somewhat similar to a matrix of synaptic weights between two layers in artificial neural networks since elements of the matrix define the contribution from one layer's activations to the next layer. There are, however, differences. The relation matrix between item and compositional memories is sparse and binary where one in the intersection of ith row and jth column means that ith HD vector is a component of jth HD vector. The sparsity is expected because usually a compositional HD vector is formed only by few components.

Another useful way of representing the same information is in the form of an executable program, which takes the atomic items as variables and produces terminal symbols of the compositional memory. For example in Figure 3 the program in Algorithm 1 computes the list of items in the compositional memory for $Arg1 = a$, $Arg2 = b$, $Arg3 = c$, $Arg4 = a \oplus b$, $Arg5 = \text{Sh}(c, 1)$:

```
Data: HD vectors from item memory (atomic, bindings and permutations)
Passed as arguments Arg1, Arg2, Arg3, Arg4 , Arg5, ...
Result: Computed terminal VSA symbols of the compositional memory
 tmp1=[Arg2, Arg3] //list of atomic vectors repeating in severalcompositions
tmp2=[Arg1]+tmp1 //list of atomic vectors repeating in several compositions
A=BUNDLE(tmp2)
B=BUNDLE([Arg4]+tmp1)
C=BUNDLE([Arg1,Arg2,Arg5])
D=BUNDLE(tmp1+[A]) //this compositional item contains compositional item A
...
return [A, B, C, D, ...]
```

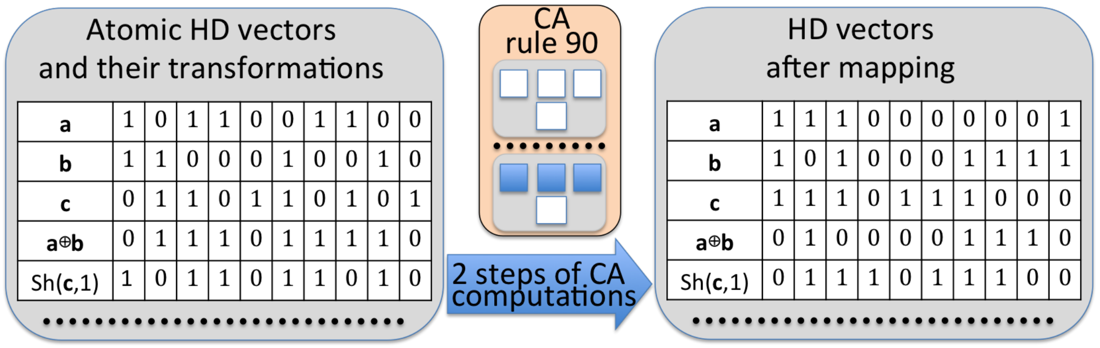
Algorithm 1: An example of a program describing the relationships between items of the VSA-based BICA memory.

Now, given the mechanism for describing the relationships between VSA-based memory the process for the replication of the entire memory into a new unique basis is straightforward and consists of two steps:

1. Mapping of the content of the entire item memory to an orthogonal basis with the help of CA_{90} computations. This is illustrated in Figure 4.

2. Replicating the compositional memory (either with the help of the relation matrix or executing a program) for new values of items in the item memory.

Thus, as the result of the memory replication procedure a new memory of HD vectors is created. All items of this new memory are dissimilar to all items of the original memory, however, the relations between atomic items and within compositional memory are identical.



a	1	0	1	1	0	0	1	1	0	0
b	1	1	0	0	0	1	0	0	1	0
c	0	1	1	0	1	1	0	1	0	1
a⊕b	0	1	1	1	0	1	1	1	1	0
Sh(**c**,1)	1	0	1	1	0	1	1	0	1	0

a	1	1	1	0	0	0	0	0	0	1
b	1	0	1	0	0	0	1	1	1	1
c	1	1	1	0	1	1	1	0	0	0
a⊕b	0	1	0	0	0	0	1	1	1	0
Sh(**c**,1)	0	1	1	1	0	1	1	1	0	0

Figure 4: Mapping the first part of the memory with two steps of cellular automata computations.

It is important to note that each step of CA_{90} computations produces a new mapping dissimilar to all other mappings. In other words, multiple replicas with unique basis can be produced by repeating the replication procedure for different number of CA_{90} execution steps.

4.1 Simulations

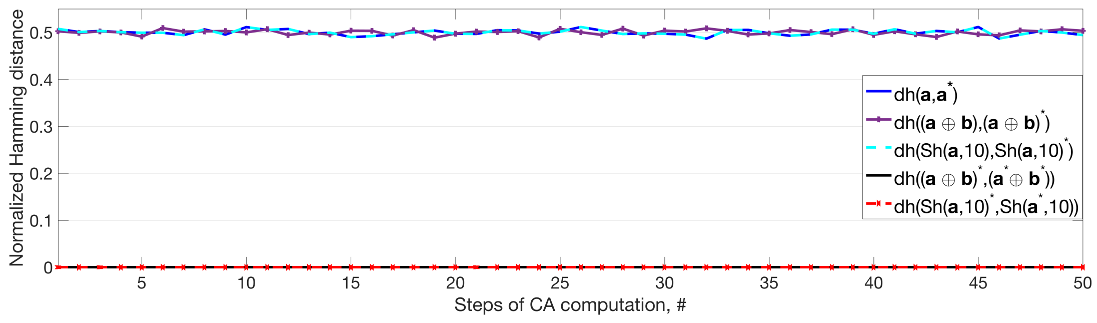


Figure 5: Similarity between the source HD vectors and mapped HD vectors for 50 computational steps with rule 90.

Properties of CA_{90} computations are illustrated in Figure 5 which plots the results of simulations of a toy memory. The origin item memory consists of four HD vectors: two atomic HD vectors ($\mathbf{a}$ and $\mathbf{b}$), one binding ($\mathbf{a} \oplus \mathbf{b}$), and one permutation (Sh($\mathbf{a}$, 10)). The source item memory was mapped to 50 dissimilar memories using 50 computational steps of CA_{90}: one step per mapping. The figure depicts five different curves, plotting normalized Hamming distance versus the number of computational steps. It is clear that the curves are grouped in two regions: in the region of the absolute similarity ($d_h = 0$) and in the region of the dissimilarity ($d_h \approx 0.5$). Three curves in the dissimilarity region show normalized Hamming distance between three HD vectors in the source memory ($\mathbf{a}$, $\mathbf{a} \oplus \mathbf{b}$, and Sh($\mathbf{a}$, 10)) and their mapped versions (denoted as $\mathbf{a}^*$, $(\mathbf{a} \oplus \mathbf{b})^*$, and Sh$(\mathbf{a}, 10)^*$) after i steps of computations. Two curves in the absolute similarity region show normalized Hamming distance between two mapped HD vectors ($(\mathbf{a} \oplus \mathbf{b})^*$ and Sh$(\mathbf{a}, 10)^*$) and results of corresponding operations (binding and permutation) performed with mapped atomic HD vectors ($(\mathbf{a}^* \oplus \mathbf{b}^*)$ and $\mathrm{Sh}(\mathbf{a}^*, 10)$). The results of the simulation confirm that the mapped memories are dissimilar to the origin item memory (and to each other) but the relationships between HD vector in the mapped memories are preserved.

5 Conclusions

This paper considered a problem of replicating the associative memory of a technical BICA system into a unique basis of atomic representations. The importance of the problem is motivated both from neural-philosophical perspective (no two brains are alike) and from the point of view of security considerations in practical BICA applications. The major challenge is to preserve all previously learned relationships while having all items (symbols) completely unrelated to the memory-origin. For BICA systems employing associative memory based on the principles of Vector Symbolic Architectures, we proposed a pipline for flexible creation of multiple replicas sutisfying the uniqueness property. The core of the replication pipeline is the mechanism for describing relationships between the item and compositional memory (a relation matrix or an executable program) and a procedure for mapping the content of the item memory to the new unique basis using computation of CA_{90}. To the best of our knowledge this has never been done before and opens a way towards secure practical applications of BICA systems.

Acknowledgements. This study is supported in part by the Swedish Research Council (grant no. 2015-04677). The authors thank Ozgur Yilmaz for fruitful discussions during BICA2016 on the usage of cellular automata in the scope of hyperdimensional computing, which inspired the current work and Niklas Karvonen for general discussions on cellular automata.

References

[1] E. S Finn, X. Shen, D. Scheinost, M. D. Rosenberg, J. Huang, M. M. Chun, X. Papademetris, and R. T. Constable. Functional connectome fingerprinting: identifying individuals using patterns of brain connectivity. *Minds and Machines*, 18:1664–1671, 2015.

[2] N. Bostrom. Quantity of experience: brain-duplication and degrees of consciousness. *Nature Neuroscience*, 16:185–200, 2006.

[3] S.D. Levy and R. Gayler. Vector symbolic architectures: A new building material for artificial general intelligence. In *Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, pages 414–418, 2008.

[4] C. Eliasmith. *How to Build a Brain.* Oxford University Press, 2013.

[5] P. Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009.

[6] S. I. Gallant and T. W. Okaywe. Representing objects, relations, and sequences. *Neural Computation*, 25(8):2038–2078, 2013.

[7] T. A. Plate. Holographic reduced representations. *Neural Networks, IEEE Transactions on*, 6(3):623–641, 1995.

[8] D. Aerts, M. Czachor, and B. De Moor. Geometric analogue of holographic reduced representation. *Journal of Mathematical Psychology*, 53:389–398, 2009.

[9] D. A. Rachkovskij. Representation and Processing of Structures with Binary Sparse Distributed Codes. *Knowledge and Data Engineering, IEEE Transactions on*, 3(2):261–276, 2001.

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

[11] P. Kanerva. What We Mean When We Say “Whats the Dollar of Mexico?”: Prototypes and Mapping in Concept Space. In , editor, *AAAI Fall Symposium. Quantum Informatics for Cognitive, Social, and Semantic Processes*, pages 2–6. , 2010.

[12] O. Yilmaz. Machine Learning Using Cellular Automata Based Feature Expansion and Reservoir Computing. *Journal of Cellular Automata*, 10(5-6):435–472, 2015.

[13] O. Yilmaz. Symbolic Computation Using Cellular Automata-Based Hyperdimensional Computing. *Neural Computation*, 27(12):2661–2692, 2015.

[14] D. Kleyko, S. Khan, E. Osipov, and S. P. Yong. Modality Classification of Medical Images with Distributed Representations based on Cellular Automata Reservoir Computing. In *IEEE International Symposium on Biomedical Imaging*, pages 1–4, 2017.