LETTER Communicated by Richard Rohwer

Symbolic Computation Using Cellular Automata-Based Hyperdimensional Computing

Ozgur Yilmaz
ozguryilmazresearch.net
Turgut Ozal University, Department of Computer Engineering,
Ankara 06010, Turkey

This letter introduces a novel framework of reservoir computing that is capable of both connectionist machine intelligence and symbolic computation. A cellular automaton is used as the reservoir of dynamical systems. Input is randomly projected onto the initial conditions of automaton cells, and nonlinear computation is performed on the input via application of a rule in the automaton for a period of time. The evolution of the automaton creates a space-time volume of the automaton state space, and it is used as the reservoir. The proposed framework is shown to be capable of long-term memory, and it requires orders of magnitude less computation compared to echo state networks. As the focus of the letter, we suggest that binary reservoir feature vectors can be combined using Boolean operations as in hyperdimensional computing, paving a direct way for concept building and symbolic processing. To demonstrate the capability of the proposed system, we make analogies directly on image data by asking, What is the automobile of air?

1 Introduction

Many real-life problems in artificial intelligence require the system to remember previous input. Recurrent neural networks (RNN) are powerful tools of machine learning with memory. For this reason, they have become one of the top choices for modeling dynamical systems. In this letter, we propose a novel recurrent computation framework that is analogous to echo state networks (ESN) (see Figure 1b) but with significantly lower computational complexity. The proposed algorithm uses cellular automata in reservoir Computing (RC) architecture (see Figure 1c) and is capable of long-term memory and connectionist computation. Moreover, the binary nature of the feature space and the additivity of the cellular automaton rules enable Boolean logic, facilitating symbolic computation directly on cellular automata reservoir. Our study is a cross-fertilization of cellular automata, reservoir computing, and hyperdimensional computing frameworks (see Figure 1a).

Neural Computation **27, 2661–2692** (2015) © 2015 Massachusetts Institute of Technology
doi:10.1162/NECO_a_00787

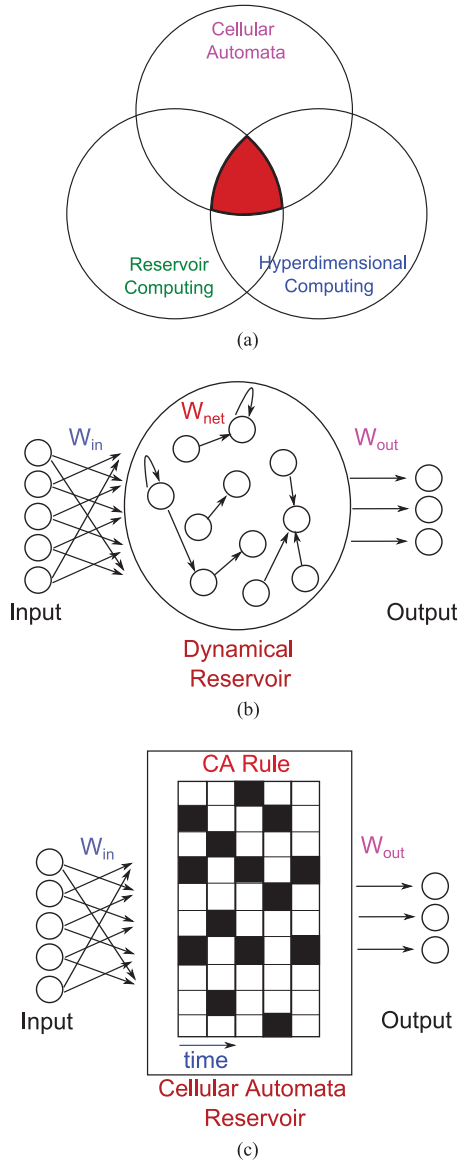

Figure 1: (a) Our work is a cross-fertilization of cellular automata, reservoir computing, and hyperdimensional computing frameworks. (b) In classical reservoir computing, data are projected on a randomly generated RNN called a reservoir, and the activities of the network are harvested. (c) In a cellular automata reservoir, data are projected onto a cellular automaton instead of a neural network.

In the following sections, we review reservoir computing, cellular automata, and neurosymbolic computation. Then we explain the contribution of our study to the field.

1.1 Reservoir Computing. Recurrent neural networks (RNNs) are connectionist computational models that use distributed representation with nonlinear units. Information in RNNs is propagated and processed in time through the states of its hidden units, which make them appropriate tools for sequential information processing.

RNNs are known to be Turing-complete computational tools (Siegelmann & Sontag, 1995) and universal approximators of dynamical systems (Funahashi & Nakamura, 1993). They are particularly appealing for problems that require remembering long-range statistical relationships such as speech, natural language processing, video processing, and financial data analysis.

Despite their immense potential as universal computers, problems arise in training RNNs due to the inherent difficulty of learning long-term dependencies (Hochreiter, 1991; Bengio, Simard, & Frasconi, 1994; Hochreiter & Schmidhuber, 1997) and convergence issues (Doya, 1992). Recent advances, however, suggest promising approaches in overcoming these issues, such as using a special memory unit that allows learning long-term dependencies in long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997; Schmidhuber, Wierstra, Gagliolo, & Gomez, 2007), using a more capable second-order optimization procedure during gradient descent (Martens & Sutskever, 2011), using a reservoir of coupled oscillators (Jaeger, 2001; Maass, Natschläger, & Markram, 2002), or emulating the recurrent computation in neural network framework (Yilmaz, 2015a).

Reservoir computing (also known as echo state networks or liquid state machines) alleviates the problem of training a recurrent network by using a randomly connected dynamical reservoir of coupled oscillators operating at the edge of chaos. It is claimed that many of these types of dynamical systems possess high computational power (Bertschinger & Natschläger, 2004; Legenstein & Maass, 2007). In the reservoir computing approach, due to rich dynamics already provided by the reservoir, there is no need to train many recurrent layers, and learning takes place only at the output (readout) layer. This simplification enables using RNNs in complicated tasks that require memory for long-range (both spatial and temporal) statistical relationships.

The essential feature for proper functioning of randomly connected networks is called echo state property (Jaeger, 2001). In networks with this property, the effect of previous state and previous input dissipates gradually in the network without getting amplified. In classical echo state networks, the network is generated randomly and sparsely, considering the spectral radius requirements of the weight matrix. Although spectral radius constraint ensures the effectiveness of the network to some extent, it does not

indicate anything about the short-term memory or computational capacity of the network. The knowledge of this capacity is essential in designing the reservoir for the given task.

The reservoir is expected to operate at the edge of chaos because the dynamical systems are shown to exhibit high computational power at this mode (Bertschinger & Natschläger, 2004; Legenstein & Maass, 2007). High memory capacity is also shown for reservoirs at the edge of chaos. A Lyapunov exponent is a measure of edge-of-chaos operation in a dynamical system, and it can be empirically computed for a reservoir network (Legenstein & Maass, 2007). However, this computation is not trivial or automatic and needs expert intervention (Lukoševičius & Jaeger, 2009).

It is empirically shown that there is an optimum Lyapunov exponent of the reservoir network related to the amount of memory needed for the task (Verstraeten, Schrauwen, dHaene, & Stroobandt, 2007). Thus, fine-tuning the connections in the reservoir for learning the optimal connections that lead to optimal Lyapunov exponent is crucial to obtain good performance with the reservoir. Many types of learning methods have been proposed for tuning the reservoir connections (see Lukoševičius & Jaeger, 2009, for a review); however, optimization procedures on the weight matrix are prone to get stuck at a local optimum due to high curvature in the weight space.

1.2 Cellular Automata. A cellular automaton is a discrete computational model consisting of a regular grid of cells, each in one of a finite number of states (see Figure 1c). The state of an individual cell evolves in time according to a fixed rule, depending on the current state and the state of its neighbors. The information presented as the initial states of a grid of cells is processed in the state transitions of cellular automaton, and computation is typically very local. Some of the cellular automata rules are proven to be computationally universal, capable of simulating a Turing machine (Cook, 2004).

The rules of cellular automata are classified according to their behavior: attractor, oscillating, chaotic, and edge of chaos (Wolfram, 2002). Turing complete rules are generally associated with the last class (rule 110, Conways game of life). Lyapunov exponents of a cellular automaton can be computed, and they are shown to be good indicators of the computational power of the automata (Baetens & De Baets, 2010). A spectrum of Lyapunov exponent values can be achieved using different cellular automata rules. Therefore, a dynamical system with specific memory capacity (i.e., Lyapunov exponent value) can be constructed by using a corresponding cellular automaton (or a hybrid automaton (Sipper, Tomassini, & Capcarrere, 1998).

Cellular automata have been previously used for associative memory and classification tasks. Tzionas, Tsalides, and Thanailakis (1994) proposed a cellular automaton-based classification algorithm. Their algorithm clusters 2D data using cellular automata, creating boundaries between different

seeds in the 2D lattice. The partitioned 2D space creates a geometrical structure resembling a Voronoi diagram. Different data points belonging to the same class fall into the same island in the Voronoi structure, hence are attracted to the same basin. The clustering property of cellular automata is exploited in a family of approaches, using rules that form attractors in lattice space (Chady & Poli, 1998; Ganguly, Sikdar, Deutsch, Canright, & Chaudhuri, 2003). The attractor dynamics of cellular automata (CA) resembles Hopfield network architecture (Hopfield, 1982). The time evolution of cellular automata has a rich computational representation, especially for edge-of-chaos dynamics, but this is not exploited if the presented data are classified according to the converged basin in 2D space. To alleviate this shortcoming, the proposed algorithm in this letter exploits the entire time evolution of the CA and uses the states as the reservoir of nonlinear computation.

1.3 Symbolic Computation on Neural Representations. Uniting the expressive power of mathematical logic and pattern recognition capability of distributed representations (e.g., neural networks) has been an open question for decades, although several successful theories have been proposed (Laird, Newell, & Rosenbloom, 1987; Anderson & Lebiere, 2014; Pollack, 1990; Van der Velde & De Kamps, 2006; Bader, Hitzler, & Hölldobler, 2008). The grand challenge is systematically tackled by a large group of scientists, and the designs are most of the time inspired by cortical models (Samsonovich, 2012). There are many theories, but a dominant solution has not yet been proposed. The difficulty arises due to the very different mathematical nature of logical reasoning and dynamical systems theory. We conjecture that combining connectionist and symbolic processing requires commonalizing the representation of data and knowledge. Recently Jaeger (2014) proposed a novel framework, called Conceptors, based on reservoir computing architecture. Conceptors are linear operators learned from the activities of the reservoir neurons, and they can be combined by elementary logical operators, which enables them to form symbolic representations of the neural activities and build semantic hierarchies. In a similar vein, Mikolov, Sutskever, Chen, Corrado, and Dean (2013) successfully used neural network representations of words (language modeling) for analogical reasoning.

Kanerva (2009) introduced hyperdimensional computing that uses high-dimensional random binary vectors for representing objects and predicates for symbolic manipulation and inference. The general family of the methods, called reduced representations or vector symbolic architectures, and detailed introductions can be found in Plate (2003) and Levy and Gayler (2008). In this approach, high dimensionality and randomness enable binding and grouping operations (for extensions, see Snaider, 2012; Snaider & Franklin, 2012) that are essential for one shot learning, analogy making, and hierarchical concept building. Most recently, Gallant and Okaywe

(2013) introduced random matrices to this context and extended the binding and quoting operations. The two basic mathematical tools of reduced representations are vector addition and XOR.

In this letter, we borrow these tools of hyperdimensional computing framework and build a semantically more meaningful representation by removing the randomness and replacing it with the cellular automata computation. Although our approach shares some commonalities with Conceptors (Jaeger, 2014) with regard to reservoir computing, the core theory and main mathematical tools are very different from our hyperdimensional computing-based approach. This is mainly due to the use of real-valued neurons and their noninteger activities as the reservoir, as opposed to binary activation of cellular automata cells. In that case, building a linearly operable representation from the neural activity requires matrix operations, which is missing in hyperdimensional computation.

1.4 Contributions. We provide a low-computational-complexity method for implementing reservoir computing–based recurrent computation, using cellular automata. Cellular automata replace the echo state neural networks. This approach provides both theoretical and practical advantages over classical neuron-based reservoir computing. The classification performance of the CA feature is examined on a CIFAR 10 data set (Krizhevsky & Hinton, 2009) in section 4.3, in which we show that CA features exceed the performance of RBF kernel with a fraction of the computation. The computational complexity of the framework is orders of magnitude lower than echo state network–based reservoir computing approaches (Yilmaz, 2015b). However, the main focus of the letter is not classification or connectionist computation performance in general, but the symbolic processing capabilities of the proposed system. In this letter, we show that the framework has great potential for symbolic processing such that the cellular automata feature space can directly be combined by Boolean operations as in hyperdimensional computing; hence, they can represent concepts and form a hierarchy of semantic interpretations. We demonstrate this capability by making analogies directly on images (see Tables 3 and 4). In the next section, we give the details of the algorithm and then provide results on simulations and experiments that demonstrate our contributions. (The details of each algorithm and experiment are given in the appendix.)

2 Algorithm

In our reservoir computing method, data are passed on a cellular automaton instead of an echo state network and the nonlinear dynamics of cellular automaton provide the necessary projection of the input data onto an expressive and discriminative space. Compared to classical neuron-based reservoir computing, the reservoir design is trivial: cellular automaton rule selection. Utilization of edge-of-chaos automaton rules ensures

Figure 2: General framework for cellular automata–based reservoir computing.

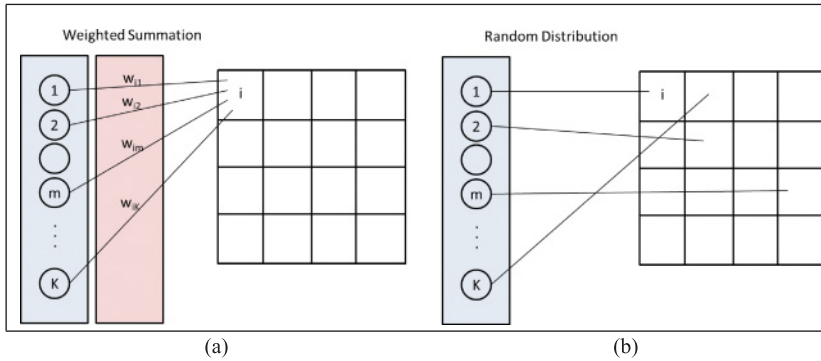

Figure 3: Two options for encoding the input into cellular automaton initial states. (a) Each cell receives a weighted sum of the input dimensions. (b) Each feature dimension is randomly mapped onto the cellular automaton cells. This last option is adopted for all the experiments in the letter.

Turing-complete computation in the reservoir, which is not guaranteed in classical reservoir computing approaches.

The algorithmic flow of our method is shown in Figure 2. The reservoir computing system receives the input data. First, the encoding stage translates the input into the initial states of a 1D or multidimensional cellular automaton (2D is shown as an example; details are given below where we explain the encoding stage). Second, in the reservoir computing stage, the cellular automaton rules are executed for a fixed period of iterations (I) to evolve the initial states. The evolution of the cellular automaton is recorded such that at each time step, a snapshot of the whole states in the cellular automaton is vectorized and concatenated. This output is a projection of the input onto a nonlinear cellular automaton state space. Then the cellular automaton output is used for further processing according to the task (e.g., classification, compression, clustering).

In the encoding stage, there are two proposed options depending on the input data:

1. For nonbinary input data, each cell of the cellular automaton might receive weighted input from every feature dimension of the input (see Figure 3a). The weighted sum is then binarized for each cell. In

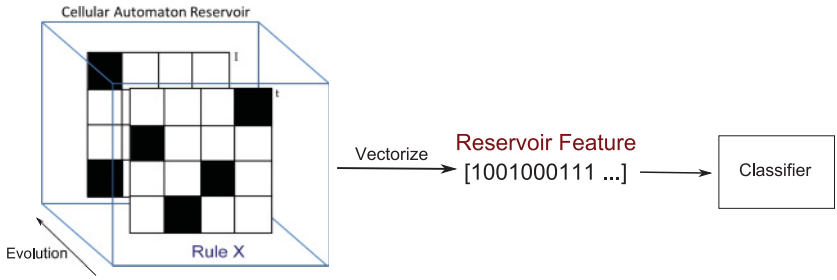

Figure 4: Cellular automaton (CA) reservoir, the space-time volume of the automaton evolution using rule X. The evolution of the CA is vectorized, and it is used as the reservoir feature for classification.

this option, instead of receiving input from the whole set of feature dimensions, a single cell can receive input from a subset of feature dimensions. In that case, the weight vector for a cell is sparse, and a subspace of the input is processed by specific cells. In general, the weights can be set randomly as in echo state networks.

2. For binary input data, each feature dimension can randomly be mapped onto the cells of the cellular automaton (see Figure 3b). The size of the CA should follow the input feature dimension.

For simplicity, we adopted the second option throughout the letter.

After encoding with the second option, suppose that the cellular automaton is initialized with the vector ${A_0}^{P_1}$, in which P_1 corresponds to a permutation of raw input data. Then cellular automata evolution is computed using a prespecified rule, Z (see Figure 4):

$$
\begin{aligned}
{A_1}^{P_1} &= Z({A_0}^{P_1}),\\
{A_2}^{P_1} &= Z({A_1}^{P_1}),\\
&\dots\\
{A_I}^{P_1} &= Z({A_{I-1}}^{P_1}).
\end{aligned}
$$

We concatenate the evolution of cellular automata to obtain a reservoir for a single permutation:

$$A^{P_1} = [{A_0}^{P_1}; {A_1}^{P_1}; {A_2}^{P_1}; \dots, {A_I}^{P_1}].$$

It is experimentally observed that multiple random mappings significantly improve accuracy. There are R number of different random mappings

(i.e., separate CA reservoirs), and they are combined into a large reservoir feature vector:

$$\mathbf{A^R} = [A^{P_1}; A^{P_2}; A^{P_3}; \ldots, A^{P_R}].$$

The computation in CA takes place when cell activities due to nonzero initial values (i.e., input) mix and interact. Both prolonged evolution duration and the existence of different random mappings increase the probability of long-range interactions, hence improve computational power.

3 Pattern Recognition Using Cellular Automata Reservoir

Cellular automata have been used for pattern recognition in the literature, but we are presenting a novel perspective. We believe that the reservoir computing framework is a perfect fit for harnessing the computational power and flexibility of cellular automata. To this end, we have previously shown in an Arxiv submission (not peer reviewed) that cellular automata reservoirs perform well in pattern recognition tasks that feedforward architectures have been unable to perform (Yilmaz, 2015b). (Key derivations of the cellular automata feature space are given in an online supplement that provides a short introduction to the core theory.) Pattern recognition capability has these characteristics:

- Noiseless memorization tasks such as 5 bit and 20 bit are proposed to be problematic for feedforward architectures, also reported as the hardest of the tasks for echo state networks in Jaeger (2012). In these tasks, a sequence of binary vectors is presented; then, following a distractor period, a cue signal is given, after which the output should be the initially presented binary vectors. We have shown that a cellular automata reservoir is able to generate the output correctly for a wide variety of conditions using the fraction of the computation used by echo state networks.
- Our analysis has shown that a cellular automata reservoir holds a distributed representation of attribute statistics of the input, just as neural networks do. We demonstrated the dramatic performance improvement due to the distributed representation over local encoding. Compared to recurrent neural networks, a cellular automata reservoir not only provides flexibility in fine-tuning the amount of memory but also requires significantly less computation.
- Kernel-based methods are widely used statistical machine learning tools (Vapnik, 2000). The kernel trick enables implicit computation of instance distances (or dot products) in the expanded feature space without ever computing the expanded feature. In the context of a cellular automata reservoir, we kernelized the system by devising a shortcut computation of reservoir feature distances without cellular

automata evolution computation. The computational complexity of the cellular automata kernel is identical to the standard linear kernel, but its performance converges to nonlinear kernels. Kernelization of cellular automata reservoir bridges the gap between kernel-based machine learning and recurrent architectures.

The results on pattern recognition experiments are promising, yet more experimentation and analyses are needed in order to place the cellular automata reservoir approach among existing architectures. More specifically, we need to compare its capability with recurrent neural algorithms such as LSTM (Hochreiter & Schmidhuber, 1997, and a successful variant, Evolino: Schmidhuber et al., 2007), classical recurrent neural nets with Hessian-free optimization (Martens & Sutskever, 2011), and echo state networks (Jaeger, 2001) using a wide variety of tasks and data sets.

In this letter, we demonstrate the symbolic computation capability of the cellular automata reservoir. The starting point is the idea that hidden layer neural activities can be used as fixed-length embeddings of a wide variety of data, such as vision or language. One of the main topics of current AI research is to learn these embeddings from the data. On the other extreme, even random embeddings can be used to do symbolic computation, as in reduced representation or vector symbolic architectures. We believe that it is not a good idea to learn the embedding from the data or the task since it is expected to degrade the generalization capability for unseen data and a different task (Gallant & Okaywe, 2013). Also, it might not be necessary for performance as long as a computationally powerful feature expansion (i.e., cellular automata) is used.

Yet random embedding is not very useful if the need is to use classical statistical machine learning tools such as support vector machines. The main inference algorithm for random embedding approaches is 1 nearest-neighbor matching, also called clean-up or autoassociative memory. This shortcoming is due to the fact that randomness does not allow building a unitary model (e.g., a decision boundary, a regression weight matrix), as in statistical machine learning approaches. Suppose that a set of person versus nonperson images will be used to train a system for detecting people in images. In statistical machine learning, a function is learned from the data, which gives a label output,

$$L_i = f(A_i),$$

in which L_i is the label of the ith image, f is the learned function, and A_i is the representation of the ith image. Since the representation of the image is randomly generated in hyperdimensional computing, the decision function, f, cannot be learned. Symbolic manipulation in hyperdimensional computing, due to analogy or rule-based inference, generates vectors in the space, and these vectors fall inside a close proximity of existing

randomly created vectors. This observation is the main ingredient of 1 nearest-neighbor based retrieval. This shortcoming was spotted by Gallant and Okaywe (2013): "It is possible to improve recognition ability for bundle vectors when the vectors added together are not random." A cellular automata reservoir gives a balanced approach: skip embedding learning, but keep the potential for machine learning.

4 Symbolic Processing and Nonrandom Hyperdimensional Computing

4.1 Combining Connectionist and Symbolic Capabilities. Two relevant symbolic processing frameworks are Conceptors (Jaeger, 2014) and hyperdimensional computing (Kanerva, 2009). Conceptors use neural representation harvested via a neural reservoir, whereas hyperdimensional computing uses binary vectors. Although both approaches can be pursued for cellular automata–based reservoir computing, in this study we are exploring the expressive power of hyperdimensional representation.

Hyperdimensional computing uses random very large-sized binary vectors to represent objects, concepts, and predicates. Then appropriate binding and grouping operations are used to manipulate the vectors for hierarchical concept building, analogy making, learning from a single example, and so forth, that are hallmarks of symbolic computation. The large size of the vector provides a vast space of random vectors, two of which are always nearly orthogonal. Yet the code is robust against a distortion in the vector due to noise or imperfection in storage because after distortion, it will stay closer to the original vector than the other random vectors.

The grouping operation is normalized vector summation, and it enables forming sets of objects/concepts. Suppose we want to group two binary vectors, V_1, and V_2. We compute their element-wise sums, and the resultant vector contains 0, 1, and 2 entries. We normalize the vector by accepting the 0 entries as they are, transforming 2 entries into 1. Note that these are consistent within the two initial vectors. Then the inconsistent entries are randomly decided: 1 entries as transformed into 0 or 1. Many vectors can be combined iteratively or in a batch to form a grouped representation of the bundle. The resultant vector is similar to all the elements of the vector due to the fact that consistent entries are untouched. The elements of the set can be recovered from the reduced representation by probing with the closest item in the memory, and consecutive subtraction. Grouping is essential for defining "a part of," "contains" relationships. A+ symbol will be used for normalized summation in the following arguments.

There are two binding operations: bitwise XOR (circled plus symbol, $\oplus$) and permutation.[1] Binding operation maps (randomizes) the vector

[1] See Kanerva (2009) for the details of permutation operation as a way of doing multiplication.

to a completely different space, while preserving the distances between two vectors. As Kanerva (2009) stated, "when a set of points is mapped by multiplying with the same vector, the distances are maintained, it is like moving a constellation of points bodily into a different (and indifferent) part of the space while maintaining the relations (distances) between them. Such mappings could play a role in high-level cognitive functions such as analogy and the grammatical use of language where the relations between objects are more important than the objects themselves." This is readily observed by considering the distance between two XORed vectors, X and Y:

$$X_A = X \oplus A,$$
$$Y_A = Y \oplus A.$$

The Hamming distance between the two vectors is the total number of 1s after XOR:

$$d(X, Y) = |X \oplus Y|.$$

Then the distance between X_A and Y_A is the same with X and Y:

$$d(X_A, Y_A) = |X_A \oplus Y_A| = |A \oplus X \oplus A \oplus Y| = |X \oplus Y|.$$

The term *multiplication* is used instead of *mapping* in the original paper. However, this is a deliberate choice because *multiplication* has a more specific meaning in this context if we consider the distributivity of XOR over addition,

$$A \oplus [X + Y + Z] = [A \oplus X + A \oplus Y + A \oplus Z],$$

in which brackets mean normalization after multiple-term summation. This property resembles the distributivity of multiplication over addition and is useful for the application of the framework for symbolic manipulation.

Following are a few representative examples to demonstrate the expressive power of hyperdimensional computing:

1. We can represent pairs of objects by multiplication. $O_{A,B} = A \oplus B$ where A and B are two object vectors.
2. A triplet is a relationship between two objects, defined by a predicate. This can similarly be formed by $T_{A,P,B} = A \oplus P \oplus B$. These types of triplet relationships are successfully used for information extraction in large knowledge bases (Dong et al., 2014).

3. A composite object can be built by binding with attribute representation and summation. For a composite object C,

$$C = X \oplus A_1 + Y \oplus A_2 + Z \oplus A_3,$$

where A_1, A_2, and A_3 are vectors for attributes and X, Y, and Z are the values of the attributes for a specific composite object.
4. A value of an attribute for a composite object can be substituted by multiplication. Suppose we have assignment $X \oplus A_1$. Then we can substitute A_1 with B_1 by $(X \oplus A_1) \oplus (A_1 \oplus B_1) = X \oplus B_1$. It is equivalent to say that A_1 and B_1 are analogous. This property is essential for analogy making.
5. We can define rules of inference by binding and summation operations. Suppose we have a rule stating, "If x is the mother of y and y is the father of z, then x is the grandmother of z."[2] Define atomic relationships:

$$M_{xy} = M_1 \oplus X + M_2 \oplus Y,$$
$$F_{yz} = F_1 \oplus Y + M_2 \oplus Z,$$
$$G_{xz} = G_1 \oplus X + G_2 \oplus Z.$$

Then the rule is

$$R_{xyz} = G_{xz} \oplus (M_{xy} + F_{yz}).$$

Given the knowledge base "Anna is the mother of Bill" and "Bill is the father of Cid," we can infer the grandmother relationship by applying rule R_{xyz}:

$$M_{ab} = M_1 \oplus A + M_2 \oplus B,$$
$$F_{bc} = F_1 \oplus B + M_2 \oplus C,$$
$$G'_{ac} = R_{xyz} \oplus (M_{ab} + F_{bc}),$$

where vector G'_{ac} is expected to be very similar to G_{ac}, which says, "Anna is the grandmother of Cid." Note that the if-then rules represented by hyperdimensional computing can only be if-and-only-if logical statements because operations used to represent the rules are symmetric.
6. A sequence of objects can be compactly represented by convoluted permutation. Suppose we have a sequence $ABCD$. Then the vector for sequence is defined as

$$S = \Pi(\Pi(\Pi(\Pi A) + B) + C) + D,$$

[2]The example is adapted from Kanerva (2009).

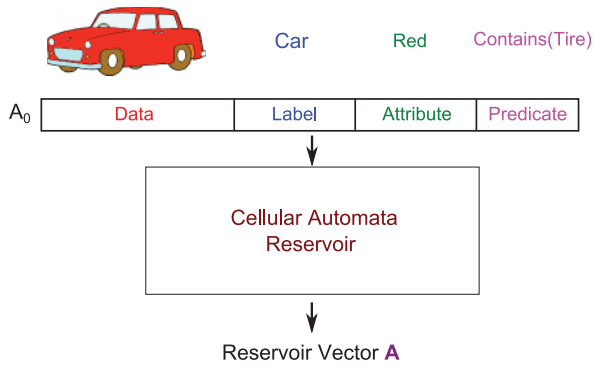

Figure 5: The hyperdimensional vector generation architecture. Each object can be defined by its data: image, label, attributes and related predicates.

in which permutation is used for binding and + is normalized summation. Then, the stored sequence can be generated in a similar way objects in a set are recovered.

Without losing the expressive power of classical hyperdimensional computing, we introduce cellular automata to the framework. In our approach, we use a binary cellular automata reservoir vector as the representation of objects and predicates instead of random vectors for symbolic computation. There are two major advantages of this approach over random binary vector generation:

1. Reservoir vectors enable connectionist pattern recognition and statistical machine learning (as demonstrated in Yilmaz, 2015b), while random vectors are mainly tailored for symbolic computation.
2. The composition and modification of objects can be achieved in a semantically more meaningful way. The semantic similarity of the two data instances can be preserved in the reservoir hyperdimensional vector representation, but there is no straightforward mechanism for this in the classical hyperdimensional computing framework. This is because a new object needs a new random vector. It is possible to distort an existing random vector to be assigned for a new semantically similar object, but how much distortion is going to be applied?

We are envisioning a structure for a cellular automaton initial vector (see Figure 5) that includes the data (e.g., image, text, or even multimodal data such as ImageAndText), object label (e.g., car), object attributes (e.g., red), and related predicates (e.g., Contains). A reservoir feature is obtained by cellular automaton reservoir expansion to be used for connectionist and symbolic computation. If we want to create another reservoir vector that

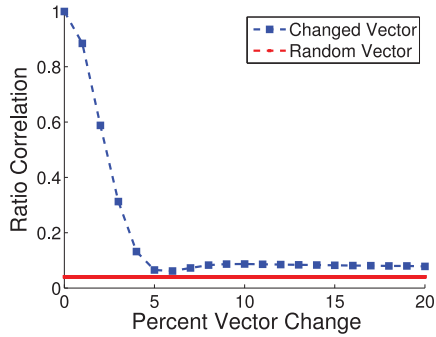

Figure 6: The simulation results that investigate the effect of changing a vector on the reservoir feature. See the text for details.

uses the same data (image) but with a modified label (e.g., vehicle), we only change the label section of the initial vector. Then the two reservoir vectors will be similar, as they should be, because they differ very little in terms of definition. The amount of reservoir vector similarity is proportional to the initial vector similarity, yet the cellular automata evolution provides a "distributed similarity," unlike a random distortion in the vector.

How much difference in the initial vector can be tolerated? This is related to the Lyapunov exponent of the cellular automaton, and it is demonstrated for rule 90 (details of the rule are given in the following section) in Figure 6. The ratio of correlation between the original reservoir vector and the changed reservoir vector is given with respect to the percentage of change in the initial vector (A_0). Experiments indicate that with a change of up to 4% in initial vector definition, the modified or distorted reservoir vector is inside the semantic space of the original vector.

Cellular automata reservoir vector representation for linear CA rules resembles the matrix binding of additive terms (MBAT) architecture of Gallant and Okaywe (2013) or the sequence coding of Kanerva (2009). In that approach, a sequence of objects is embedded into a fixed length vector by continuous matrix multiplication (see item 6 of the representative examples above). In addition to matrix operations; cellular automata enable a wide spectrum of nonlinear operations by using different rules. (See p. 9 in the online supplementary document for more details about the MBAT comparison.) Also, our framework gives a direct way of vector computation for nontemporal tasks. Thus, it can be considered as a generalization of MBAT architecture.

4.2 Symbolism for Linear Cellular Automata Rules. Very few of the cellular automaton rules are additive (e.g., rules 90 and 150), most of them being nonlinear. In this section, we analyze the symbolism for linear CA

rules. When the reservoir is created using these linear rules, the system is identical to a classical knowledge base with propositional logic inference. Symbols can be manipulated by the logical operators in a knowledge base; similarly, reservoir vectors can be manipulated by the corresponding logical operators. Thus, the working space can be shifted from symbols to reservoir vectors without losing expressive power. To make the distinction clear, let us rephrase that any cellular automaton rule can be used for reservoir expansion, and hyperdimensional computing capabilities will not change for different CA rules. However, linear rules are providing computational and theoretical opportunities, and they will be explored here.

For the additive cellular automata rule, 90, the evolution is governed by XOR operation. Suppose that cellular automaton is initialized with vector A_0, which holds the attributes of the raw data. The cellular automaton activity at time 1 is given by

$$A_1 = \Pi_1 A_0 \oplus \Pi_{-1} A_0,$$

where Π_1 and Π_{-1} are matrices $+1$ and -1 bit shift operations, and $\oplus$ is bitwise XOR.

Similarly,

$$\begin{aligned}
A_2 &= \Pi_2 A_0 \oplus \Pi_{-2} A_0, \\
A_3 &= \Pi_3 A_0 \oplus \Pi_{-3} A_0 \oplus \Pi_1 A_0 \oplus \Pi_{-1} A_0, \\
A_4 &= \Pi_4 A_0 \oplus \Pi_{-4} A_0, \\
A_5 &= \Pi_5 A_0 \oplus \Pi_{-5} A_0 \oplus \Pi_3 A_0 \oplus \Pi_{-3} A_0, \\
&\dots
\end{aligned}$$

Linear CA rules show additive behavior: the evolution for different initial conditions can be computed independently, and then the results are combined by simply adding (Chaudhuri, 1997; Wolfram, 2002). For example, rule 90 is additive under exclusive or (XOR) operation such that when two separate initial conditions are combined by XOR (shown by $\oplus$), their subsequent evolution can also be combined by XOR. Suppose we have two separate initial conditions, A_0 and B_0, and the single time step state evolution function is given by ϕ. Then:

$$\phi(A_0 \oplus B_0) = \phi(A_0) \oplus \phi(B_0).$$

Although the combination logic for rules 150 and 22 (because it simulates rule 90) can also be derived, we will focus on rule 90.

Suppose we have two separate inputs, A, B. Let us assume that the nonzero entries in the input (i.e., the initial states of the CA) represent the

existence of categorical objects, as in 5 bit/20 bit tasks. We compute the reservoir by applying the rule (i.e., 90) for a period of time steps and concatenating the state space; call them C_A and C_B. We are interested in a new concept by combining the two inputs: $A \vee B$. This new concept should represent the union of objects, existing separately in A and B; thus, it is more abstract. Due to the binary categorical indicator nature of the input feature space, the definition of logical combination rules is straightforward. We define OR operation by computing the reservoir of $A \vee B$:

$$OR(A, B) = C_{A \vee B} = C_A \oplus C_B \oplus C_{A \wedge B} = C_A \oplus C_{B-A}.$$

The representation of the new concept obtained by the union on existing concepts can be computed on the cellular automata reservoir feature space via XOR operation, which is equivalent to the addition operation on Galois Field, Z_2. OR is a grouping operation that is achieved by normalized summation in a classical hyperdimensional computing framework.

If the pattern is already stored in a concept, a repetitive addition does not cause a change,

$$C_{(A \vee B) \vee B} = C_{A \vee B} \oplus C_{\mathbf{0}} = C_{A \vee B},$$

and this is essential for incremental storage (Jaeger, 2014).

An AND operation will generate a concept that consists of categories that coexist in both A and B:[3]

$$AND(A, B) = C_{A \wedge B} = C_A \oplus C_{A-B}.$$

For the application of AND/OR operations, the initial vector of the CA reservoir needs to be saved for all logical entities in order to extract C_{A-B} and C_{B-A}, yet this is not a practical issue.

XOR operation is straightforward:

$$XOR(A, B) = C_A \oplus C_B.$$

In classical hyperdimensional computing, normalized summation can be used as AND in rule formation (item 5 in section 4.1). However, the real meaning of the operation as the intersection of concepts cannot be used; thus, AND does not really exist. This major advantage of the CA framework is due to the semantically meaningful computation of vectors via cellular automata evolution instead of random generation. In fact, in linear CA symbolism, it is as if operations are manipulating the initial vector that holds logical variables. The semantics of the initial vectors are preserved

[3]Derived using De Morgan's rule and experimentally verified.

after every logical operation on the reservoir due to additivity, for example, when the reservoirs are OR'ed, the resultant vector is identical to a reservoir computed with OR'ed initial conditions. Therefore, the system is expressively equivalent to a propositional logic language. Overall it has great expressive power; the availability of XOR and AND forms the whole Z_2 field, and it is possible to represent any logic obtainable by (OR, AND), with the additional benefit of algebraic operations. Yet the reservoir expansion also provides a means for statistical machine learning as demonstrated in the previous sections. The symbolic computation in linear CA rule system resembles hard Conceptors in Jaeger (2014), although the XOR operation seems missing in the latter.

Despite the fact that linear CA rules provide a very capable symbolic system, nonlinear rules are expected to perform better on machine learning tasks. Therefore, there seems to be a trade-off between the two goals in terms of favored CA rules.

4.3 Experiments on Cellular Automata Symbolic Computation. It should be noted that the symbolic computation property of the linear cellular automaton framework is applicable when the nonzero feature attributes of the CA initial states represent the existence of a predefined object/concept.[4] This is not an issue for object labels, attributes, or predicates (see Figure 5), but sensory data are not necessarily categorical. Having said that, any sensory data space can be transformed into categorical indicator space by using a feedforward neural network in the front end (see Figure 7).[5] This approach is parallel to the three-stage proposal by Gallant and Okaywe (2013), in which the preprocessing stage provides the required feature expansion.

We have previously shown that binarization of the hidden layer activities of a feedforward network is not very detrimental for classification purposes Yilmaz, Ozsarac, Gunay, and Ozkan (2015). For an image, the binary representation of the hidden layer activities holds an indicator for the existence of Gabor-like corner and edge features, hence provides the means for the required categorical indicator space. The proposed perspective on feedforward networks generalizes the local binary pattern (LBP) analysis (Ojala, Pietikainen, & Harwood, 1994). This capability can be generalized to domains other than images and higher-level descriptions of objects.

In order to test the performance of CA features on binarized hidden layer activities, we run experiments on the CIFAR 10 data set. We used the first 500 training and test images and obtained single layer hidden neuron

[4]Binary coding for initial conditions of the CA is another option, but logical operations are meaningless in this case.

[5]Another option for categorical transformation is quantization of the data (or the frequency spectrum—DFT or Hadamard—of the data), then encoding the quanta in binary, which might not be practical for high-dynamic-range problems.

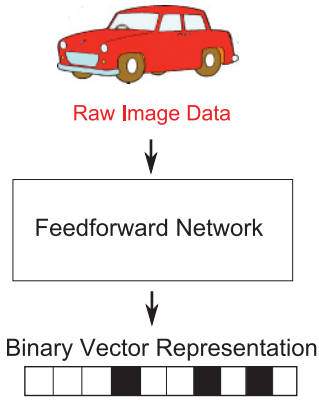

Figure 7: Binary representation for data that are inherently nonbinary (e.g., an image) can be obtained by binarizing the feedforward network representation of the data.

Table 1: Experiments on a CIFAR 10 Data Set (Subset).

	Linear	RBF
Real neuron	34.6	35.2
Binary neuron	37.6	39.6
CA feature (max/avg)	41.8/39	-

Notes: The classification performance for real and binary neural representation is given for linear and RBF kernels. The CA feature is computed on binary neuron activities, and the linear kernel performance on the CA feature (maximum and average are given) is equivalent to RBF kernel performance on binary neurons.

representation using the algorithm in Coates, Ng, and Lee (2011) (200 different receptive fields; receptive fields are the size of six pixels). The neural activities are binarized according to a threshold, and, on average, 22% of the neurons fired with the selected threshold. After binarization of neural activities, CA features can be computed (rule 90, $R = 4$ and $I = 4$) on the binary representation, as explained in section 2. We trained linear and RBF kernel SVM classifiers Chang and Lin (2011) on both real and binary neural activities, as well as CA features that are computed on the binary neuron activities. The parameters of the SVM classifiers are optimized using a coarse-to-fine grid search, and CA feature experiment is repeated many times due to randomized permutation. The results are given in Table 1. It is observed that the binary neuron shows superior performance over the real neuron, which has to be investigated further. What is more significant

Table 2: Experiments on the CIFAR 10 Data Set (Subset).

Hidden Layer Conceptor	CA Conceptor
28.3	32.5

Note: The classification performance of Conceptors that are computed from binary hidden-layer activities and cellular automata features for $R = 16$ and $I = 16$.

is that the CA feature on the linear classifier exceeds (best random permutation) the RBF kernel nonlinear classifier. Yet the computational complexity is much lower (the expansion is $R \times I = 16$), and it has potential for improvement via hybridization and multilayer architecture (see section 7). However, many more experiments with larger data sets are needed to accurately evaluate the performance limits of the CA features.

We also tested the classification performance of CA conceptors as in Jaeger (2014). We formed a separate Conceptor for each class using binary neural representation (rule 110, $R = 16$ and $I = 16$; 50 samples for each class) of CIFAR training data and vector addition defined in Snaider (2012). Then we tested the classification performance of the Conceptors on test data in such a way that each test data CA feature vector is associated with the closest (maximum inner product) Conceptor. The results are given in Table 2. It is observed that classification with the CA Conceptor outperforms the classification with a hidden layer Conceptor, which supports the idea that CA expansion enhances the symbolic computation. More important, the Conceptor classification provides a much more flexible framework, discussed in Jaeger (2014), where addition or deletion of data and classes is straightforward and logical operations are enabled on the semantic space.

In order to demonstrate the power of enabled logical operation, we will use analogy making, which is crucial for generalization of what is already learned. We tested the capability of our symbolic system using images. The example given here follows, "What Is the Dollar of Mexico?" in Kanerva (2009). However, in the original example, sensory data (i.e., image) are not considered because there is no straightforward way to introduce them into the hyperdimensional computing framework. The benefit of using nonrandom binary vectors is obvious in this context.

We formed two new concepts, called Land and Air:

$$Land = Animal \oplus Horse + Vehicle \oplus Automobile,$$

$$Air = Animal \oplus Bird + Vehicle \oplus Airplane.$$

In these two concepts, the CA features of Horse and Bird images are used to bind with the Animal filler, and the CA features of Automobile and

Table 3: Analogy-Making Experiment on CIFAR 10 Data Set (Subset).

Hidden Layer Feature	CA Feature: Rule 110	Rule 90	Rule 22	Rule 182
21.3	67.2	46.1	48.9	63.4

Notes: The accuracy of the analogy is given for binary hidden layer neuron representation and CA features for several CA rules. See the text for details.

Airplane images are used to bind with the Vehicle filler (50 training images for each; rule 110, R and I are both 16). The Animal and Vehicle fields are represented by two random vectors (22% nonzero elements) with the same size as the CA features. Multiplication is performed by XOR ($\oplus$) operation, and vector summation is again identical to Snaider (2012). The products Land and Air are also CA feature vectors, and they represent the merged concept of observed animals and vehicles in Land and Air respectively. We can ask the analogical question, "What is the Automobile of Air?" (*AoA* for short). The answer is given by

$$AoA = Automobile \oplus Land \oplus Air.$$

AoA is a CA feature vector and expected to be very similar to the Airplane Conceptor. We tested the analogical accuracy using unseen Automobile test images (50 in total), computing their CA feature vectors followed by *AoA* inference, then finding the closest Conceptor class to the *AoA* vector (maximum inner product). It is expected to be the Airplane class. The result of this experiment is given in Table 3. The analogy on CA features is 67% accurate, whereas if the binary hidden layer activity is used instead of CA features (corresponds to R and I equal to 1), the analogy is only 21% accurate. This result clearly demonstrates the benefit of CA feature expansion on symbolic computation, even more than the classification task given in Table 2. Additionally, we show that the Turing complete rule 110 performs significantly better than some other viable rules (i.e., 90, 22), whereas rule 182 approaches the performance of rule 110.

The devised analogy implicitly assumes that the Automobile concept is already encoded in the concept of Land. What if we ask, "What is the Truck of Air?"? Although Truck images are not used in building the Land concept due to the similarity of Truck and Automobile concepts, we might still get good analogies. The results on these second-order analogies are contrasted in Table 4. Automobile and Horse ("What is the Horse of Air?" the answer should be Bird) are first-order analogies, and they result in comparably superior performance as expected, but second-order analogies are much higher than chance level (i.e., 10%).

These analogies are performed strictly on sensory data—images. Given an image, the system is able to retrieve a set of relevant images that are

Table 4: Another Analogy-Making Experiment on the CIFAR 10 Data Set (Subset).

Automobile	Horse	Truck	Deer	Ship	Frog
67.2	58.2	38.6	38.9	31.5	24.4

Notes: The accuracy of the analogy is given for first (given in bold) and second-order analogies. See the text for details.

linked through a logical statement. A very small number of training data are used, yet we can infer conceptual relationships between images surprisingly accurately. However, again it should be emphasized that this is only a single experiment with a very limited analogical scope, and more experiments are needed to understand the limits of the proposed architecture. Also, there is room for performance improvement since we are reporting accuracy around 60%.

It is possible to build much more complicated concepts using hierarchies. For example, Land and Air are types of environments and can be used as fillers in the Environment field. Ontologies are helpful to narrow down the set of required concepts for attaining a satisfactory description of the world. Other modalities such as text data are also of great interest (see Mikolov et al., 2013 and Pennington, Socher, & Manning, 2014, for state-of-the-art studies, as well as information fusion on multiple modalities, such as Image and text).

5 General Architecture for Connectionist-Symbolic Machine Intelligence

In this section, we devise a holistic artificial intelligence architecture. Although it is not supported with experimental work yet, we believe it is appropriate to provide our opinion because it is built on previous studies considered to be solid in the literature. We propose that reservoir computing and hyperdimensional computing can work together to achieve simultaneous connectionist and symbolic intelligence, as shown in Figure 8.

Through reservoir computing-based machine learning, it is possible to achieve these partial estimations:

1. Estimate object label when data are given (classification)
2. Estimate object attributes when data are given (regression)
3. Estimate object label when attributes are given (descriptive models)
4. Generate data when attributes and label are given (generative statistical model)

More significant, when these partial estimations are performed, the initial sensory processing is done and memory retrieval is enabled. The

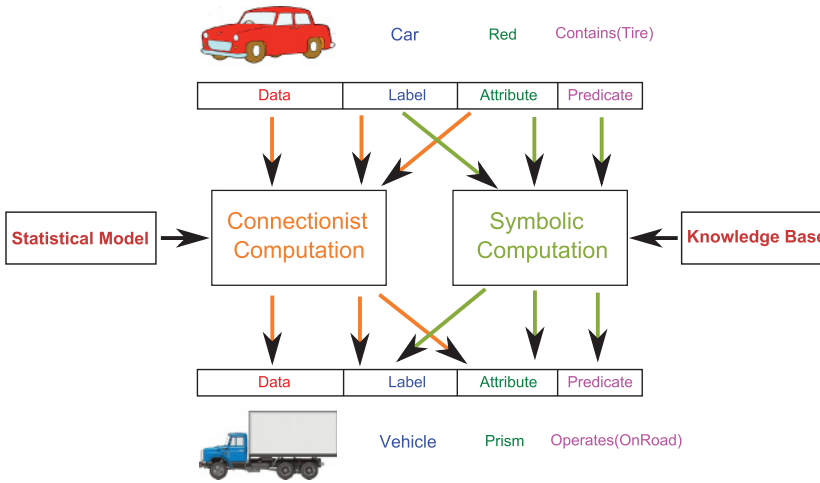

Figure 8: Combining connectionist and symbolic processing through reservoir and hyperdimensional computing. See the text for details.

data + *label* + *attribute* vector is projected on the CA reservoir space, and it is stored as sparse distributed memory (SDM; Kanerva, 1993). Then, SDM-based retrieval works in tandem with machine learning–based partial estimations as an autoassociative memory to fill-in missing data and improve precision and accuracy (see also Yilmaz, 2015a, for a low-complexity option). We conjecture that this retrieval is the essence of recurrent computation in the human cortex.

Using the power of hyperdimensional computing, we can use labels, attributes, and predicates to achieve

- Composite object description via attributes and predicates
- Analogy making
- Rule-based inference

These are partial symbolic estimations. After performing the possible logical inferences, the *label* + *attribute* + *predicate* vector is projected on the CA reservoir space and again stored as SDM. Similarly, SDM-based retrieval is enabled that fills in and regularizes incomplete knowledge.

As the data structure is filled in with extracted knowledge through simultaneous processing of the two streams, such as more detailed attributes for an object (color, shape, material, size, symmetry), a class label hierarchy (vehicle-car-sedan), or an intricate relationship of predicates (Contains(Car,Tire), isPartOf(Tire,Vehicle), isBelow(Tire,Window)), the vector becomes a rich description of the data. Both the statistical model and

knowledge base are dynamic entities in this picture: they should adapt and evolve to include the learned knowledge.

Data, labels, and attributes are projected onto reservoir feature space for sensory processing; labels, attributes, and predicates are projected onto reservoir feature space for nontraditional (i.e., hyperdimensional) symbolic processing. Thus, both low-level data and high-level knowledge occupy the same space. After partial estimations and retrievals, we have a more complete *data* + *label* + *attribute* + *predicate* vector. This is a blackboard of a cognitive session. This whole vector can be projected onto the CA reservoir feature space and saved as a semantic memory (Squire, 1992).

In the proposed architecture, there is cross-talk between connectionist and symbolic computation via labels and attributes. The framework offers a potential for holistic artificial intelligence, where we can use the results of pattern recognition on logical inference, and vice versa.

We can give a few examples for possible use scenarios:

1. An object-scene relationship in an image can be handled using a high-order language, where we can define complicated logical rules: if we detect a computer object and a nearby desk object, and then we infer that it is an office scene if there is not a car, tree, or building objects in the image.
2. We can infer hidden states. In an image, if we detect a face object but cannot detect a human object and we detect a chair object nearby the face object, then the person is sitting.
3. We can start a search for possible objects. If we classify the scene as a street, start a search for cars and humans.
4. We can detect parts of an object. If we detect a car in the scene, we can search for its parts, such as tires, windows, and lights, through a Contains predicate. If we cannot detect its parts, we can raise a flag using an Occluded predicate. We can infer its orientation by detected parts: if two tires and two windows but no lights are detected, for example, the orientation is sideways. If we infer the orientation, then we can repeat the detection and search with this knowledge, possibly with a more refined classifier.
5. Sensory feedforward flow forms beliefs about the external world while goal-based feedback can cause priors, and this is mainly achieved by SDM retrievals. We can use logical statements to define goals, and these can modify labels and attributes—hence, gate sensory perception.

Simultaneous connectionist and symbolic computation is a huge claim. It not only requires a complete system where domain-dependent ontologies, procedural memory, and production execution are included (Anderson & Lebiere, 2014), but also deserves a thorough examination, which is planned as a future work.

6 Computational Complexity

There are two major savings of cellular automata framework compared to classical echo state networks:

1. Cellular automaton evolution is governed by bitwise operations instead of floating-point multiplication.
2. Since the reservoir feature vector is binary, matrix multiplication needed in the linear classification or regression can be replaced by summation.

Multiplication in echo state network is replaced with bitwise logic (e.g., XOR for rule 90), and multiplication in classification/regression is replaced with summation. Overall, multiplication is completely avoided in both reservoir computation and classifier and regression stages, which makes the CA framework especially suitable for FPGA hardware implementations.

For symbolic processing, there is no additional computation for a CA framework, and reservoir outputs can be combined directly using logical rules as in hyperdimensional computing. However, Conceptors (Jaeger, 2014) built on ESN require correlation matrix computation and matrix multiplication of large matrices for each input. As an example, for the 20-bit task $T_0 = 200$, 1760 million floating-point operations are needed for correlation matrix computation. Then there is a matrix inversion (2000×2000 size, 68 million operations) and matrix multiplication (two 2000×2000 size matrices, 16,000 million operations) to obtain the Conceptor matrix. All these computations (about 18 billion) are avoided in our framework.

7 Discussion

We provide a novel reservoir-computing framework that is capable of long-term memory and symbolic processing, which requires significantly less computation compared to echo state networks Yilmaz (2015b).[6] In the proposed reservoir computing approach, data are passed on a cellular automaton instead of an echo state network (Figure 1), and, similar to echo state networks with sparse connections, the computation is local (only two neighbors in 1D, implying extreme locality) in the cellular automaton space. Several theoretical advantages of the cellular automata framework compared to echo state networks are mentioned, with their practical benefits. Cellular automata are easier to analyze, have insurance on Turing completeness, and allow Boolean logic as well as algebra on a Galois field. From the CA side, the computation performed in cellular automata can be conceptualized at many levels (Mitchell, Schuster, & Grams, 1996; James, Mitchell, & Dasz,

[6]Detailed comparison with other RNN algorithms are not provided, but the computational complexity argument seems to be generally valid.

1998), but our main proposition is that reservoir computing is a very good fit for harnessing the potential of cellular automaton computation.

Some of the best-performing cellular automaton rules are additive. How does extremely local, additive, and bitwise computation give a surprisingly good performance in a pathological machine learning task? This is a question that needs further examination. We suggest that if a dynamical system has universal computation capability, it can be used for difficult tasks that require recurrent processing once it is properly used. The trick that worked in the proposed framework is multiple random projections of the inputs that contributed to the probability of long-range interactions. Nevertheless, we need more experimental evidence for the true limits and machine learning performance of the cellular automaton reservoir framework.

Along with the pattern recognition capabilities of cellular automaton–based reservoir computing, hyperdimensional computing framework enables symbolic processing. Due to the binary categorical indicator nature of the representation, the rules that make up the knowledge base and feature representation of the data that make up the statistical model occupy the same space, which is essential for combining connectionist and symbolic capabilities. It is possible to make analogies, form hierarchies of concepts, and apply logical rules on the reservoir feature vectors.

In the experiments (see Tables 1 and 2), we showed that CA-based symbolic computation approaches the performance of statistical machine learning–based classification (i.e., SVM). Yet we have not searched for any type of optimization or even tried the symbolic system with very large vectors, so there is room for improvement. Additionally, Conceptor-based classification provides a much more flexible framework for data and class maintenance, with the added benefit of logical operations and queries. To illustrate the logical query, we have shown the capability of the system to make analogies on image data. We asked, "What is the Automobile of Air?" after building Land and Air concepts based on the images of Horse, Automobile (Land), Bird, and Airplane (Air). The correct answer is Airplane, and the system infers this relationship with 67% accuracy, with only 50 training images per class. These preliminary results are encouraging, yet we need to find more cases to test the symbolic computation capabilities of the framework—specifically on rule-based logical inference and compositional concept hierarchy building.

Neural network data embeddings (Kiros, Salakhutdinov, & Zemel, 2014; Mikolov, Sutskever, Chen, Corrado, and Dean, 2013) are an alternative to our approach, in which representation suitable for logical manipulation is learned from the data using gradient descent. Although these promising approaches are showing state-of-the-art results, they are bound to suffer from the dilemma of "no free lunch" because the representation is data specific. The other extreme is random embeddings adopted in hyperdimensional computing and reduced vector representation approaches. Although randomness maximizes the orthogonality of vectors and optimizes the effective

use of the space, it does not allow statistical machine learning or semantically meaningful modifications on existing vectors. Our approach lies in the middle: it does not create random vectors, and thus can manipulate existing vectors and use machine learning, but it does not learn the representation from the data and therefore is less prone to overfitting as well as to the dilemma of no free lunch. In addition, CA Reservoir seems to be orders of magnitude faster than neural network counterparts, as discussed in section 6.

A future direction for the potential application of the framework is data fusion. The proposed data representation (see Figure 5) can be used for fusion on multiple levels: the sensor level using data and label and the knowledge level using attribute and predicates. We can concatenate multiple objects one after another such as an image object and a text (i.e., image caption) object. Then the reservoir will compute interobject statistics, capturing cross-modal interactions. Although there is an exciting advancement in simultaneous image-text learning in the neural network literature (Kiros et al., 2014), it is strictly in the realm of statistical machine learning, being incapable of representing high-level knowledge bases and logical rules.

As a future study, we plan to test the framework on large data sets for tasks such as language modeling, music and handwriting prediction, and computer vision.

Appendix: Details of Experiments

This appendix provides the details of the experiments performed in the letter. The Matlab codes for all experiments are found at ozguryilmazresearch .net.

A.1 Cellular Automata Reservoir. We have a binary data vector of size N. We permute the data vector, compute cellular automata states for a fixed period of time, and concatenate to get CA reservoir feature vector. The computation is done in a nested-for loop:

```
for permutation = 1 to R
   Retrieve a random permutation
   Permute data
   for iteration = 1 to I
      Compute next state of CA
      Store in a matrix
   end for loop
   Store CA evolution in a matrix
end
```

Concatenate the CA evolution matrix to get a vector of size $N \times I \times R$. The only difference in Game of Life is that we map the input data vector onto a 2D square grid of suitable size during permutation.

A.2 Experiments on Cellular Automata Symbolic Computation. In order to test the performance of CA features on binarized hidden layer activities, we ran experiments on a CIFAR 10 data set. We used the first 500 training and test images. We obtained single-layer hidden neuron representation using the algorithm in Coates, Ng, and Lee (2011). The code is provided by Coates on the Web. First, 200 different receptive fields of size 6 pixels are learned using k-means on 1 million randomly cropped image patches. Then, each 6-by-6 image region is sparsely encoded by the similarity to 200 receptive fields. The similarity measure is a real number, but we need a binary representation in order to run CA rules. The neural activities are binarized according to a threshold, and, on average, 22% of the neurons fired with the selected threshold. After binarization of neural activities, CA features can be computed (rule 90, $R = 4$ and $I = 4$) on the binary representation, as explained in section 2 and detailed in section A.1. We trained linear and RBF kernel SVM classifiers (Chang & Lin, 2011) on both real and binary neural activities, as well as CA features that are computed on the binary neuron activities. The parameters of the SVM classifiers are optimized using a coarse-to-fine grid search, and the CA feature experiment is repeated 20 times due to randomized permutation.

We also tested the classification performance of CA Conceptors as in Jaeger (2014). We formed a separate Conceptor for each class using the CA feature vectors, derived from binary neural representation as explained above. Rule 110 is run for $R = 16$, and $I = 16$, and a CA feature vector is obtained for each image; 50 samples are used for each class of CIFAR training data. Denote F_{ij} for jth sample of ith class.

Vector addition, defined in Snaider (2012) is used to form the Conceptors of each class:

$$\mathbf{T_i} = sign\left(\sum_{j=1}^{50} \frac{F_{ij} - 0.5}{0.5}\right).$$

Then we tested the classification performance of the Conceptors on test data, such that each test data CA feature vector (F_j) is associated with the closest Conceptor using max inner product with the Conceptor,

$$c_j = \underset{i}{\operatorname{argmax}}\, F_j T_i,$$

where c_j is the class decision on the jth test data instance. The runs are repeated for 20 trials due to randomness at CA permutation, and the verage accuracy is reported.

In the analogy experiments, Animal and Vehicle fields are randomly created binary vectors (22% nonzero elements). These two vectors are used

to form the Land and Air concepts using the equation in the corresponding section.

Acknowledgments

This research is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) Career Grant 114E554 and Turgut Ozal University BAP Grant 006-10-2013. I thank Lenovo Group Ltd. Turkey Division, specifically Cagdas Ekinci and Alpay Erturkmen, for their support in this research.

References

Anderson, J. R., & Lebiere, C. J. (2014). *The atomic components of thought*. New York: Psychology Press.

Bader, S., Hitzler, P., & Hölldobler, S. (2008). Connectionist model generation: A first-order approach. *Neurocomputing*, *71*, 2420–2432.

Baetens, J. M., & De Baets, B. (2010). Phenomenological study of irregular cellular automata based on Lyapunov exponents and Jacobians. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *20*, 033112.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*, 157–166.

Bertschinger, N., & Natschläger, T. (2004). Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, *16*, 1413–1436.

Chady, M., & Poli, R. (1998). *Evolution of cellular-automaton-based associative memories*. New York: Springer.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*, 27:1–27:27. http://www.csie.ntu.edu.tw/cjlin/libsvm

Chaudhuri, P. P. (1997). *Additive cellular automata: Theory and applications*. Volume 1. New York: Wiley.

Coates, A., Ng, A. Y., & Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics* (pp. 215–223). Washington, DC: SPARC.

Cook, M. (2004). Universality in elementary cellular automata. *Complex Systems*, *15*, 1–40.

Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., ... Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 601–610). New York: ACM.

Doya, K. (1992). Bifurcations in the learning of recurrent neural networks 3. In *Proceedings of the 1992 IEEE Symposium on Circuits and Systems* (Vol. 3, p. 17). Piscataway, NJ: IEEE.

Funahashi, K.-i., & Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, *6*, 801–806.

Gallant, S. I., & Okaywe, T. W. (2013). Representing objects, relations, and sequences. *Neural Computation*, *25*, 2038–2078.

Ganguly, N., Sikdar, B. K., Deutsch, A., Canright, G., & Chaudhuri, P. P. (2003). *A survey on cellular automata* (Technical Report). Dresden: Dresden University of Technology.

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen netzen*. Master's thesis, Institut fur Informatik, Technische Universitat, Munich.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*, 2554–2558.

Jaeger, H. (2001). *The echo state approach to analysing and training recurrent neural networks–with an erratum note* (GMD Technical Report 148). Bonn: German National Research Center for Information Technology.

Jaeger, H. (2012). *Long short-term memory in echo state networks: Details of a simulation study* (Technical Report 27). Bremen: Jacobs Univesity.

Jaeger, H. (2014). *Controlling recurrent neural networks by conceptors*. arXiv preprint arXiv:1403.3369.

James, P., Mitchell, M., & Dasz, R. (1998). *The evolutionary design of collective computation in cellular automata*. (Technical Report). Santa Fe, NM: Santa Fe Instituto.

Kanerva, P. (1993). Sparse distributed memory and related models. In M. W. Hassoun & K. F. Cheung (Eds.), *Associative neural memories* (pp. 50–76). New York: Oxford University Press.

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

Kiros, R., Salakhutdinov, R., & Zemel, R. S. (2014). *Unifying visual-semantic embeddings with multimodal neural language models*. arXiv preprint arXiv:1411.2539.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images* (Technical Report). Toronto: Computer Science Department, University of Toronto.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Legenstein, R., & Maass, W. (2007). Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, *20*, 323–334.

Levy, S. D., & Gayler, R. (2008). 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* (pp. 414–418). Amsterdam: IOS Press.

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*, 127–149.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*, 2531–2560.

Martens, J., & Sutskever, I. (2011). Learning recurrent neural networks with Hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning* (pp. 1033–1040). Madison, WI: Omnipress.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *26* (pp. 3111–3119). Red Hook, NY: Curran.

Mitchell, M., Schuster, H. G., & Grams, T. (1996). Computation in cellular automata: A selected review. *Nonstandard Computation* (pp. 95–140). Weinheim: VCH Verlagsgesellschaft.

Ojala, T., Pietikainen, M., & Harwood, D. (1994). Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In *Pattern Recognition, 1994. Vol. 1—Conference A: Proceedings of the 12th IAPR International Conference on Computer Vision and Image Processing* (pp. 582–585). Piscataway, NJ: IEEE.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, *12*, (pp. 1532–1543). Stroudsburg, PA: Association for Computational Linguistics.

Plate, T. A. (2003). *Holographic reduced representation: Distributed representation for cognitive structures*. Stanford, CA: CSLI Publications.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, *46*, 77–105.

Samsonovich, A. V. (2012). On a roadmap for the BICA challenge. *Biologically Inspired Cognitive Architectures*, *1*, 100–107.

Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F. (2007). Training recurrent networks by Evolino. *Neural Computation*, *19*, 757–779.

Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, *50*, 132–150.

Sipper, M., Tomassini, M., & Capcarrere, M. S. (1998). Evolving asynchronous and scalable non-uniform cellular automata. In D. W. Pearson, N. C. Steele, & R. Albrecht (Eds.), *Artificial Neural Nets and Genetic Algorithms* (pp. 66–70). New York: Springer.

Snaider, J. (2012). *Integer sparse distributed memory and modular composite representation*. PhD diss., University of Memphis.

Snaider, J., & Franklin, S. (2012). Extended sparse distributed memory and sequence storage. *Cognitive Computation*, *4*, 172–180.

Squire, L. (1992). Declarative and nondeclarative memory: Multiple brain systems supporting learning and memory. *Journal of Cognitive Neuroscience*, *4*, 232–243.

Tzionas, P. G., Tsalides, P. G., & Thanailakis, A. (1994). A new, cellular automaton-based, nearest neighbor pattern classifier and its vlsi implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (vol. 2, pp. 343–353). Piscataway, NJ: IEEE.

Vapnik, V. (2000). *The nature of statistical learning theory*. New York: Springer Science & Business Media.

Van der Velde, F., & De Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, *29*, 37–70.

Verstraeten, D., Schrauwen, B., dHaene, M., & Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, *20*, 391–403.

Wolfram, S. (2002). *A new kind of science*. Vol. 5. Champaign, IL: Wolfram Media.

Yilmaz, O. (2015a). *Classification of occluded objects using fast recurrent processing*. arXiv preprint arXiv:1505.01350.

Yilmaz, O. (2015b). *Connectionist-symbolic machine intelligence using cellular automata based reservoir-hyperdimensional computing*. arXiv preprint arXiv:1503.00851.

Yilmaz, O., Ozsarac, I., Gunay, O., & Ozkan, H. (2015). Cognitively inspired real-time vision core (Technical Report). http://ozguryilmazresearch.net/Publications/NeuralNetworkVisionCore_YilmazEtAl2015.pdf

Received April 14, 2015; accepted August 5, 2015.