Journal of Cellular Automata, Vol. 0, pp. 1–38
Reprints available directly from the publisher
Photocopying permitted by license only

Machine Learning Using Cellular Automata Based Feature Expansion and Reservoir Computing

OZGUR YILMAZ

Turgut Ozal University, Department of Computer Engineering, Ankara Turkey
E-mail: ozyilmaz@turgutozal.edu.tr; ozguryilmazresearch.org

Received: July 22, 2015. Accepted: August 5, 2015.

In this paper, we introduce a novel framework of cellular automata based computing that is capable of long short-term memory. 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 feature vector. The proposed framework requires orders of magnitude less computation compared to Echo State Networks. We prove that cellular automaton reservoir holds a distributed representation of attribute statistics, which provides a more effective computation than local representation. It is possible to estimate the kernel for linear cellular automata via metric learning, that enables a much more efficient distance computation in support vector machines framework.

Keywords: Cellular automata, distributed representation, metric learning, kernel methods, reservoir computing

1 INTRODUCTION

Many real life problems in artificial intelligence require the system to remember previous inputs. 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 paper we propose a novel recurrent computation framework that is analogous to Echo State Networks (ESN) (Figure 1 a), with significantly lower computational complexity.

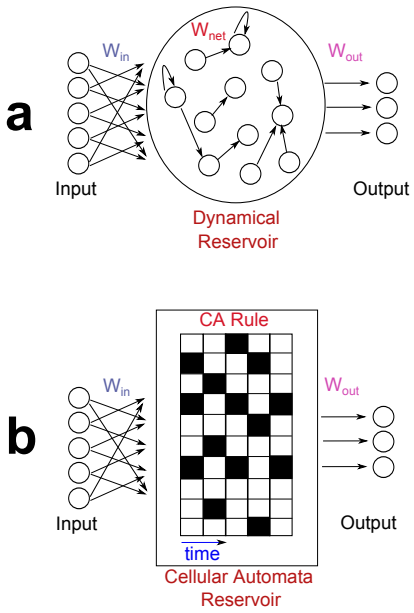

FIGURE 1
a. In classical reservoir computing, data is projected on a randomly generated recurrent neural network and the activities of the network are harvested which is called reservoir. **b.** In cellular automata reservoir, data is projected onto cellular automaton instead of a neural network.

The proposed algorithm uses cellular automata in Reservoir Computing (RC) architecture (Figure 1 b) and is capable of Long-Short-Term-Memory (LSTM).

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

1.1 Reservoir Computing

Recurrent Neural Networks are connectionist computational models that utilize 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 [58] and universal approximators of dynamical systems [25]. 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 [11, 32, 33] and convergence issues [24]. On the other hand, recent advances suggest promising approaches in overcoming these issues, such as emulating the recurrent computation in neural network framework [69] or utilizing a reservoir of coupled oscillators [36, 47].

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 type of dynamical systems possess high computational power [12, 45]. The method is closely related with reaction-diffusion computation methods [4] or collision-based computing [5], and computation in nonlinear medium [3] in general. In 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 (read-out) layer. This simplification enables usage of recurrent neural networks in complicated tasks that require memory for long-range, both spatially and temporally, statistical relationships. It can be considered in a larger family of architectures where random connections are utilized instead of trained ones, another popular feedorward algorithm is named extreme learning machines [35].[1]

The essential feature for stability of the randomly connected network is called echo state property [36]. 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. Even though spectral radius constraint ensures stability 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 proper 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 [12, 45]. High memory capacity is also shown for reservoirs at the edge of chaos. Lyapunov exponent is a measure edge of chaos operation in a dynamical system, and it can be empirically computed for a reservoir network [45]. However, this computation is not trivial or automatic, and needs expert intervention [46].

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 [65]. Thus, fine-tuning the connections in the reservoir for learning the optimal connections that lead to optimal Lyapunov exponent is very crucial to obtain good performance with the reservoir. There are many types of learning

[1] The comparison of the two approaches are given in [13].

methods proposed for tuning the reservoir connections (see [46] for a review), however optimization procedure on the weight matrix is prone to get stuck at local optimum due to high curvature in the weight space.

The input in a complex task is generated by multiple different processes, for which the dynamics and spatio-temporal correlations might be very different. One important shortcoming of the classical reservoir computing approach is its inability to deal with multiple spatio-temporal scales simultaneously. Modular reservoirs have been proposed that contain many decoupled sub-reservoirs operating in different scales, however fine tuning the sub-reservoirs according to the task is a non-trivial task.

1.2 Cellular Automata

Cellular automaton is a discrete computational model consisting of a regular grid of cells, each in one of a finite number of states (Figure 1 c). The state of an individual cell evolves in time according to a fixed rule, depending on the current state and the state of 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 [21].

The rules of cellular automata are classified [6, 67] according to their behavior: attractor, oscillating, chaotic, and edge of chaos. Turing complete rules are generally associated with the last class (rule 110, Conways game of life [2]). Lyapunov exponent of a cellular automaton can be computed and it is shown to be a good indicator of computational power of the automata [10]. 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 [59]).

Cellular automata have been previously used for associative memory and classification tasks. Tzionas *et al.* [62] 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 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. Clustering property of cellular automata is exploited in a family of approaches, using rules that form attractors in lattice space [15, 28]. The attractor dynamics of cellular automata resembles Hopfield network architectures [34]. The time evolution of the cellular automata has very 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 paper exploits the **entire** time evolution of the CA, and uses the states as the reservoir of nonlinear computation.

Another cellular based machine learning approach is cellular neural networks [18]. It has been shown that every binary cellular automata of any dimension is a special case of a cellular neural network of the same neighborhood size [19]. However, cellular neural networks impose a very specific spatial structure and they are generally implemented on specialized hardware, generally for image processing (see [39] for a recent design).

A relatively new class of cellular automata approach is Elementary Cellular Automata with Memory (ECAM) [8, 51]. The state of each cell is determined by the history of the cell using majority, minority or parity (XOR) rules. This line of work is very related with the proposed framework in this paper (section 4 on distribted representation of statistics) and a cross-talk between reservoir computing and ECAM seems essential.

1.3 Contributions

We first provide a feedorward architecture using cellular automata computation. Then we show that the proposed architecture is capable of accomplishing long-short-term-memory tasks such as the 5 bit and 20 bit noiseless memorization (Table 3), which are known to be problematic for feedforward architectures [33]. We prove that the computation performed by cellular automata produces a distributed representation of higher order attribute statistics, which gives dramatically superior performance over local representation (Table 4 and 5). Also we demonstrate that cellular automata feature expansion for linear rules can be kernelized via metric learning, and the estimated kernel significantly reduces computation in both training and test stages in support vector machines while the performance approaches nonlinear methods (Table 5).

Then we provide a low computational complexity method (Table 6) 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 computational complexity of the feedforward and recurrent framework is orders of magnitude lower than echo state network based reservoir computing approaches (Table 6). The versatility of our framework is illustrated in Figure 2. In the next section we give the details of the algorithm and then provide results on several simulations and experiments that demonstrate our contributions. The details of each the algorithms and experiments are given separately in an appendix.

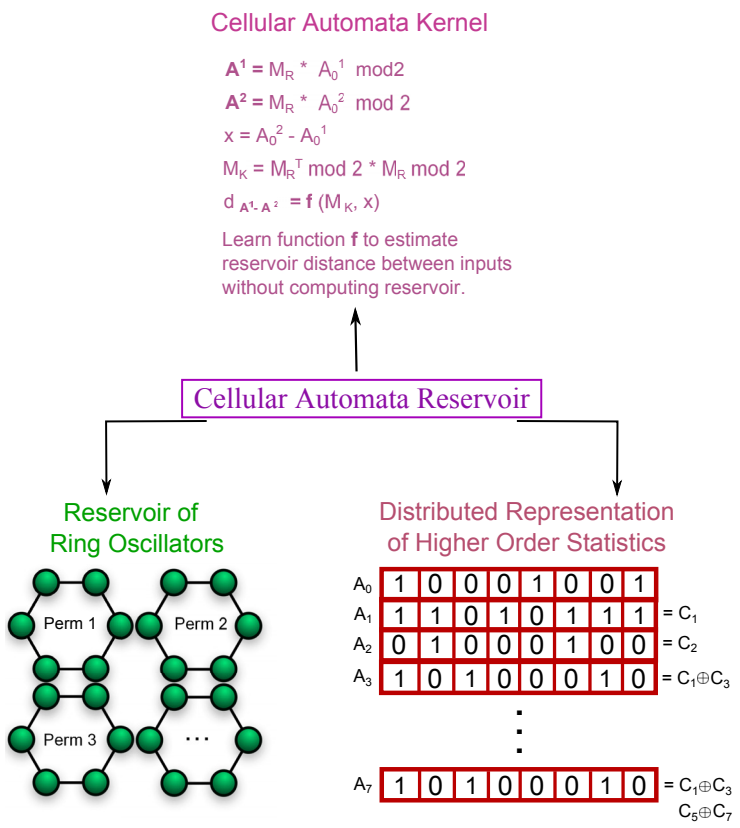

FIGURE 2
Three aspects of cellular automata based reservoir computing. **Lower Left.** Cellular automata reservoir is essentially an ensemble of ring oscillators because of one-neighbor connectivity of cellular automata. **Lower Right.** The cellular automata reservoir holds a distributed representation of high order attribute statistics (section 4), that is shown to be superior over local representation. A_0 is the initial condition for CA and it evolves in time. Whole space time evolution, A_k, is the reservoir, and it holds higher order attribute statistics given as C_k. **Upper Left.** The computation in linear cellular automaton rules can be kernelized to be used in SVM framework (section 6), where the kernel performance approaches nonlinear kernel performance with only linear computational complexity.

2 ALGORITHM

In our system, 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 feedforward or reservoir computing, the feature space design is trivial: cellular automaton rule selection.

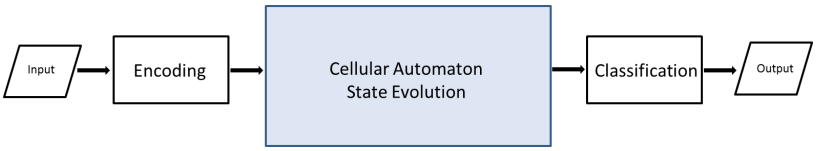

FIGURE 3
General framework for cellular automata based computing.

Utilization of 'edge of chaos' automaton rules ensures Turing complete computation in the feature space, which is not guaranteed in classical neural network approaches.

As a starting point we first provide a feedforward architecture. The analysis of the feedforward architecture gives a valuable intuition for the capabilities of the cellular automaton computation. For this purpose we look into memorization capability of the feedforward algorithm in section 3, then analyze the distributedness of the representation in section 4. We kernelize the feedforward feature space in sections 5 and 6. The neuralization of the feedforward architecture is introduced in section 7. The recurrent architecture is given in section 8, and until that section all the arguments in the paper are made for the feedforward cellular automata feature expansion. The added capability of due to recurrence is analyzed in detail, and experimental results are presented in section 8.

Algorithmic flow is shown in Figure 3. The encoding stage translates the input into the initial states of a 1D or multidimensional cellular automaton (2D is shown as an example). In cellular automaton 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 automata state space. Then the cellular automaton output is used for further processing according to the task (eg. classification, compression, clustering etc.).

In encoding stage there are two proposed options depending on the input data. **1.** For non-binary input data, each cell of cellular automaton might receive weighted input from every feature dimension of the input (Figure 4a). The weighted sum is then binarized for each cell. In 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

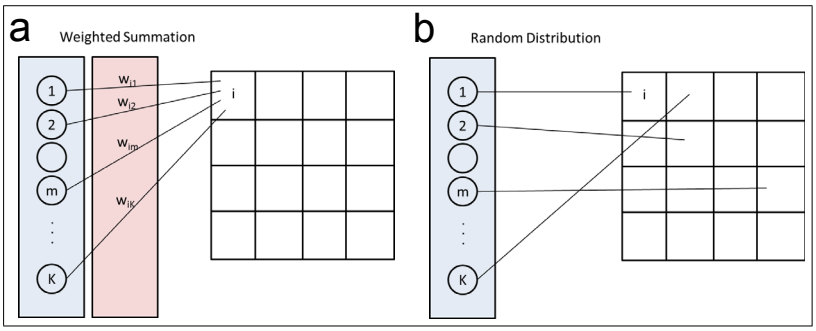

FIGURE 4
Two options for encoding the input into cellular automaton initial states. **a**. Each cell receives a weighted sum of the input dimensions. **b**. Each feature dimension is randomly mapped onto the cellular automaton cells. This last option is adopted for all the experiments in the paper.

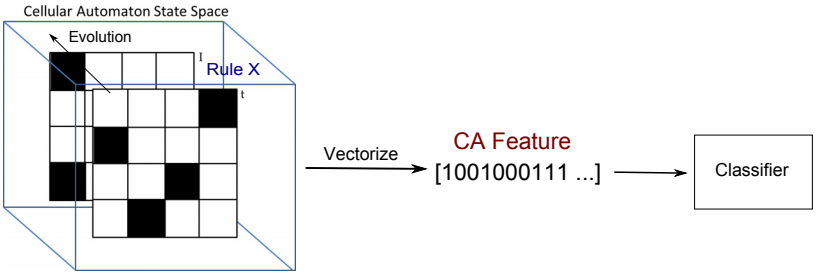

FIGURE 5
Cellular automaton (CA) feature space which is the space-time volume of the automaton evolution using Rule X. The whole evolution of the CA is vectorized and it is used as the feature vector for classification.

the cells of the cellular automaton (Figure 4b). The size of the CA should follow the input feature dimension. For the sake of simplicity we adopted the second option throughout the paper.

After encoding with second option, suppose that cellular automaton is initialized with 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 (Figure 5):

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

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

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

It is experimentally observed that multiple random mappings significantly improve accuracy. There are R number of different random mappings, i.e. separate CA feature spaces, and they are combined into a large feature vector:

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

The computation in CA takes place when cell activity due to nonzero initial values (i.e. input) mix and interact. Both prolonged evolution duration and existence of different random mappings increase the probability of long range interactions, hence improve computational power. We analyze the effect of number of evolutions and number of random mappings on computational capacity in Section 4.

3 MEMORY OF CELLULAR AUTOMATA STATE SPACE

In order to test for long-short-term-memory capability of the proposed feedforward cellular automata framework, 5 bit and 20 bit memory tasks were used. They are proposed to be problematic for feedforward architectures, also reported to be the hardest tasks for echo state networks in [37]. In these tasks, a sequence of binary vectors are presented, then following a distractor period, a cue signal is given after which the output should be the initially presented binary vectors. Input-output mapping is learned by estimating the linear regression weights via pseudo-inverse (see [37] for details). These tasks have binary input, hence it is possible to randomly map the input values onto cells as initial states (Figure 4 b). In echo state networks, a sequence is presented to the network one step at a time, while the network is remembering the previous inputs in its holistic state. In our feedforward architecture, sequence is flattened: all the sequence data are presented to the network at once (vectorization) and the whole output sequence is estimated using regression. Increasing the distractor period expands the size of the vector, and adds irrelevant data to the CA computation reducing the estimation quality.

Using a larger data portion for context is common in sequence learning and it is called sliding window method, and estimating the whole output sequence is generally utilized in hidden Markov model approaches [23], but these two were not applied together before to the best of our knowledge. Estimating the whole sequence using the whole input sequence is a much harder

$T_0 = 200$	R = 4	16	32	64	$T_0 = 1000$	R = 4	16	32	64
I = 4	100	100	100	4	I = 4	100	100	100	100
16	100	28	0	0	16	100	100	100	14
32	100	0	0	0	32	100	100	100	0
64	100	0	0	0	64	100	100	22	0

TABLE 1
Percent failed trials for 5 Bit Task, Game of Life CA, $T_0 = 200$ (Left) $T_0 = 1000$ (Right). Rows are number of iterations I, and Columns are number of random permutations.

task, and we are making our case more difficult in order to keep the comparison with literature (explained in more detail in section 8).

Both 1D elementary CA rules and 2D Game of Life CA is explored. The total computational complexity is determined by the number of different random mappings, R, (i.e. separate feature spaces, Figure 5) and the number of cell iterations I. The success criteria and regression method provided by [37] are used in the experiments.

3.1 Game of Life

5 bit task is run for distractor period T_0 200 and 1000. The percent of trials that failed is shown for various R and I combinations in Table 1. 20 bit task is run for distractor period T_0 200 and 300, and again percent failed trials is shown in Table 2. It is observed that Game of Life CA is able to solve both 5 bit and 20 bit problems and for very long distractor periods.

3.2 Elementary Cellular Automata

5 bit task ($T_0 = 200$) is used to explore the capabilities of elementary cellular automata rules. Rules 32, 160, 4, 108, 218 and 250 are unable to give meaningful results for any [R, I] combination. Rules 22, 30, 126, 150, 182, 110, 54, 62, 90, 60 are able give 0 error for some combination. Best performances are observed for rules 90, 150, 182 and 22, in decreasing order (Table 3). It is again observed that computational power is enhanced with increasing either R or I, thus multiplication of the two variables determine the overall performance.

$T_0 = 200$	R = 192	256	320	384	$T_0 = 300$	R = 192	256	320	384
I = 12	100	100	100	32	I = 12	100	100	100	60
16	100	84	12	0	16	100	100	20	0

TABLE 2
Percent failed trials for 20 Bit Task, Game of Life CA, $T_0 = 200$ (Left) $T_0 = 300$ (Right). Rows and columns are the same as above.

***Rule* 90**	**R = 8**	**16**	**32**	**64**
I = 8	100	78	12	0
16	74	4	0	0
32	4	2	0	–

***Rule* 150**	**R = 8**	**16**	**32**	**64**
I = 8	100	80	8	0
16	84	6	0	0
32	8	0	0	–

***Rule* 182**	**R = 8**	**16**	**32**	**64**
I = 8	100	82	18	0
16	92	14	0	0
32	12	0	0	–

***Rule* 22**	**R = 8**	**16**	**32**	**64**
I = 8	100	78	20	0
16	86	16	0	0
32	16	0	0	–

TABLE 3
Percent failed trials for 5 Bit Task, Elementary CA Rules, $T_0 = 200$. Rows and columns are the same as above.

Previous studies have shown that, even recurrent architectures are having hard time to accomplish these tasks, while cellular automata feature expansion is able to give zero error. Thus, the cellular automaton state space seems to offer a rich feature space, that is capable of long-short-term-memory. Interestingly, class 3 CA rules [67] which show random behavior seem to give best performance in this memorization task.[2]

4 DISTRIBUTED REPRESENTATION OF HIGHER ORDER STATISTICS

Discovering the latent structure in high dimensional data is at the core of machine learning, and capturing the statistical relationship between the data attributes is of the essence[3]. Second order statistics (covariance matrix) is shown to be very informative for a set of AI tasks [60]. However, higher order statistics are known to be required for a wide variety of problems [7, 52]. Polynomial kernels in support vector machine framework represent these statistics in a local manner [29]. The computation in cellular automata can be viewed from many perspectives using a variety of mathematical formalisms [53]. We are approaching this from a machine learning path, and trying the understand the meaning of cellular automaton evolution in terms of attribute statistics[4]. We show that linear cellular automata compute higher order statistics and hold them within its states in a distributed manner, which shows clear advantages over local representation (see Table 4 and 5).

[2]The results and arguments in the following sections will always be derived using elementary cellular automata.

[3]Sometimes this is called learning 'long range dependencies'.

[4]To calirfy the language: the initial vector A_0 of CA hold the data attributes, and the evolution of CA results in CA features.

For simplification and due to the encouraging results from the aforementioned memory experiments (Table 3), we analyzed a linear automaton, rule 90. 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,

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

See Figure 2 lower right, for visualizing the evolution.

It can be shown that by induction:

$$A_k = \Pi_k A_0 \oplus \Pi_{-k} A_0 \oplus LOT,$$

where LOT are the lower order terms, i.e. $\Pi_i A_0 \oplus \Pi_{-i} A_0$, for $i < k$.
Let us define:

$$C_k = \Pi_k A_0 \oplus \Pi_{-k} A_0,$$

which is nothing but the covariance of the attributes that are $2k$ apart from each other, which is hold at the center bit of k^{th} time step of cellular automaton evolution (Figure 6).

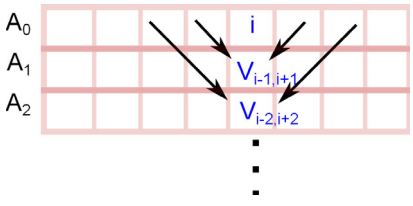

FIGURE 6
The cell value for rule 90 (i.e. $V_{i-k,i+k}$) represents covariance of initial conditions (attributes), that are $2k$ apart from each other, where k is the time step of evolution. This is demonstrated for a single column of the state evolution.

Then:

$$
\begin{aligned}
A_1 &= C_1, \\
A_2 &= C_2, \\
A_3 &= C_3 \oplus A_1, \\
A_4 &= C_4, \\
A_5 &= C_5 \oplus A_3 \oplus A_1 \\
&\dots
\end{aligned}
$$

Using induction:

$$C_k = A_k \oplus \prod_{i \in S} A_i,$$

where the set S holds lower order terms (i.e. cellular automaton time steps before k) and product symbol represents consecutive XOR operations over the defined set of cellular automaton states.

Therefore we can deduce any attribute covariance C_k, via proper application of XOR on cellular automaton state evolution, A_k. Otherwise, the cellular automaton states hold a **distributed representation of attribute higher order statistics**:

$$A_k = C_k \oplus \prod_{i \in S} C_i.$$

What is the function of multiple initial random permutations? Initial random permutation (eg. R_1) used in our framework is an extra matrix multiplication on initial vector A_0, then time evolution equation is:

$$A_1 = \Pi_1 \Pi^{R_1} A_0 \oplus \Pi_{-1} \Pi^{R_1} A_0,$$

and the two matrices can be combined into one:

$$\Pi_1^{R_1} = \Pi_1 \Pi^{R_1}.$$

Without random permutation, the i^{th} bit at time step 1 holds the covariance between $i - 1^{st}$ and $i + 1^{st}$ attributes. This is illustrated in Figure 6, represented as V:

$$A_1^i = V_{i-1,i+1},$$

***Rule* 90, $T = 200$, $I = 8$**	**16**	**32**	**64**
Distributed HOS	73	9	0
Local Covariance	94	62	11

TABLE 4
Percent failed trials for 5 Bit Task, $T_0 = 200$. Columns are number of random permutations. Distributed Higher Order Statistics (kept in CA states) and Local Covariance features are compared.

After random permutation, the same bit holds a completely different covariance between attributes j and k, determined by the permutation matrix:

$$A_1^i = V_{j,k},$$

$$R_1(i-1) = j \text{ and } R_1(i+1) = k,$$

where R_1 vector holds the random permutation of indices.

Therefore, a random permutation enables computation of a completely different set of attribute correlations. Effectively, random permutation and state evolution work in a similar fashion to include the computation of previously unseen attribute correlations. The equivalence of these two dimensions in CA framework is experimentally verified above (eg. Table 3). However, the effect of R and I is not completely symmetric for CA computation, and it is analyzed in the following sections.

Cellular automaton feature space holds distributed higher order statistics of attributes, denoted as A_k. What happens if we use covariance features, C_k? Remember that these two can be transformed to each other, however do they differ in generalization performance? We repeated 5 bit memory task experiment (T=200, I=8) using C_k features instead of A_k, and Table 4 shows the comparison. Distributed representation stored as cellular automaton states show clear performance advantage over the local covariance representation, which emphasizes the importance of cellular automata based feature expansion. In the following sections (6) we compare polynomial SVM kernel with cellular automaton kernels (Table 5), and support this argument. Although the distributed higher order statistics argument we made through derivations were for linear cellular automata rules, it can be extended for other interesting rules (eg. rule 110) by experimental studies, which is planned as a future study. As a side note, elementary cellular automata with memory (ECAM) framework provides a more powerful way of exploiting attribute statistics and it should be investigated both analytically and experimentally from reservoir computing perspective.

5 METRIC LEARNING FOR LINEAR CELLULAR AUTOMATA FEATURE SPACE

Kernel based methods are widely used statistical machine learning tools [63] (see [54] for a recent survey). The kernel trick enables implicit computation of instance distances (or dot products) in the expanded feature space, without ever computing the expanded feature. It significantly reduces the amount of computation in training and test stages, for many type of kernels, features and tasks. Kernelization is closely related with distance metric learning [42, 68], especially for nonlinear feature mappings. In the context of cellular automata feature space introduced in this paper, we would like to kernelize the system by devising a shortcut computation of cellular automata feature distances without cellular automata evolution computation. We show the kernelization for linear cellular automaton rule 90 due to its simplicity however this work can be extended for other analytically tractable rules.

Linear cellular automaton state evolution can be represented as consecutive matrix multiplication [17,27,49]. Suppose we have CA initial conditions A_0 (vector of size N), I number of state evolutions and R number of random initial projections. Then, state evolution for a single random projection is given by,

$$
\begin{aligned}
A_1 &= M_N * A_0 \; mod \; 2, \\
A_2 &= {M_N}^2 * A_0 \; mod \; 2, \\
A_3 &= {M_N}^3 * A_0 \; mod \; 2, \\
&\dots \\
A_I &= {M_N}^I * A_0 \; mod \; 2,
\end{aligned}
$$

where M_N is an $N \times N$ sparse binary matrix:

$$
M_N = \begin{bmatrix}
0 & 1 & 0 & ... & 0 & 1 \\
1 & 0 & 1 & 0 & ... & 0 \\
0 & ... & 0 & 1 & 0 & 1 \\
1 & 0 & ... & 0 & 1 & 0
\end{bmatrix}
$$

The feature space can be computed at once by concatenation of matrices:

$$
\begin{aligned}
M_I &= [M_N \; {M_N}^2 \; {M_N}^3 \; ... \; {M_N}^I], \\
\mathbf{A} &= M_I A_0 \; mod \; 2,
\end{aligned}
$$

where M_I is an $NI \times N$ matrix.

Since a random permutation of initial cellular automata configuration is another matrix multiplication, the feature space for a single random permutation is given by:

$$\mathbf{A^{R1}} = M_I \Pi^{R_1} A_0 \; mod \; 2.$$

Then the cellular automaton feature space for all the random permutations (whole CA feature space, $\mathbf{A^R}$) can be computed using a single matrix multiplication:

$$\mathbf{M} = [M_I \Pi^{R_1} \; M_I \Pi^{R_2} \; M_I \Pi^{R_3} \; ... \; M_I \Pi^{R_R}],$$
$$\mathbf{A^R} = \mathbf{M} A_0 \; mod \; 2,$$

where $\mathbf{M}$ is an $NIR \times N$ matrix, and $\mathbf{A^R}$ is a NIR sized vector.

Euclidian distance (equivalent to Hamming for binary data) between two data instances, A_0 and B_0, can be analyzed as follows:

$$\begin{aligned}
&\mathbf{d}(\mathbf{A^R}, \mathbf{B^R}) = \|\mathbf{M} A_0 \; mod \; 2 - \mathbf{M} B_0 \; mod \; 2\|, \\
&= \|\mathbf{M}(A_0 - B_0) \; mod \; 2\| && \text{because } -1^2 = 1^2, \\
&= (A_0 - B_0)^T \; \mathbf{M^T} mod \; 2 \; \times \; \mathbf{M}(A_0 - B_0) \; mod \; 2 \\
& && \text{Euclidian distance property,} \\
&= \left((A_0 - B_0)^T mod \; 2 \; \times \; \mathbf{M^T} mod \; 2\right) \; mod \; 2 \\
&\times \; (\mathbf{M} \; mod \; 2 \; \times \; (A_0 - B_0) \; mod \; 2) \; mod \; 2 && \text{Property of modulus,}
\end{aligned}$$

If we take the modulus 2 of both sides of the equation,

$$\mathbf{d}(\mathbf{A^R}, \mathbf{B^R}) \; mod \; 2 \; = (F \; mod \; 2 \; \times G \; mod \; 2 \;) \; mod \; 2 \;,$$

where F and G are new variables for the complex terms in multiplication given in the previous equation. Then we can use the property of modulus again to reach:

$$\mathbf{d}(\mathbf{A^R}, \mathbf{B^R}) \; mod \; 2 \; = (F \times G) \; mod \; 2 \;.$$

Unwrapping F and G, we get

$$\begin{aligned}
&\mathbf{d}(\mathbf{A^R}, \mathbf{B^R}) \; mod \; 2 \; = \\
&\left((A_0 - B_0)^T mod \; 2 \; \times \mathbf{M^T} mod \; 2 \times \mathbf{M} \; mod \; 2 \; \times (A_0 - B_0) \; mod \; 2\right) \; mod \; 2.
\end{aligned}$$

Let us define two new variables for simplification:

$$d_0 = (A_0 - B_0)\ mod\ 2$$

$$\mathbf{M_K} = \mathbf{M^T} mod\ 2 \times \mathbf{M}\ mod 2,$$

then we attain,

$$\mathbf{d}(\mathbf{A^R}, \mathbf{B^R})\ mod\ 2\ = {d_0}^T\ \mathbf{M_K} d_0\ mod\ 2.$$

Assuming that we can estimate the manifold via a function (g) defined on modulus 2, we get a definition of distance,

$$\mathbf{d}(\mathbf{A^R}, \mathbf{B^R}) = {d_0}^T\ \mathbf{M_K} d_0\ +\ 2 \times \mathbf{g}(d_0, \mathbf{M_K})$$

$$\mathbf{d}(\mathbf{A^R}, \mathbf{B^R}) = \mathbf{f}(d_0, \mathbf{M_K}).$$

The function f estimates the CA feature distances (i.e. $\mathbf{d}(\mathbf{A^R}, \mathbf{B^R})$) by utilizing the distance between the initial CA states of two instances (d_0), the matrix $\mathbf{M_K}$ of size $N \times N$, which is specified by the random initial projections and the number of CA iterations. A few observations:

1. ${d_0}^T d_0$ is the Euclidian (Hamming) distance between the data instances
2. ${d_0}^T\ \mathbf{M_K} d_0$ is analogous to Mahalanobis distance with covariance matrix $\mathbf{M_K}$.

Can we estimate function f for **a given dataset**? The experiments reported below (Figure 7) indicate that it is possible to very accurately estimate the distance using linear regression over features of d_0 and $\mathbf{M_K}$. But should we use the entries of d_0 and $\mathbf{M_K}$ as features, or can we devise more meaningful features? We conjecture that, the distance in CA feature space will depend on both the initial Euclidian and Mahalanobis distances of the data.

Another alternative is usage of eigenvalue matrix $\mathbf{S_K}$, derived by singular value decomposition of $\mathbf{M_K}$, for weighted Euclidian distance computation: ${d_0}^T\ \mathbf{S_K}\ d_0$. It can be computed in $\mathcal{O}(N)$ time due to the diagonal nature of $\mathbf{S_K}$. Mahalanobis distance is not only computationally more demanding but also overfitting the densities, producing unnecessarily large and elongated hyperellipsoidal components [48], hence it is a good idea to regularize it by the eigenvalue matrix [9]. Usage of the three distance features for linear regression of CA feature distance, will effectively perform that regularization. Let

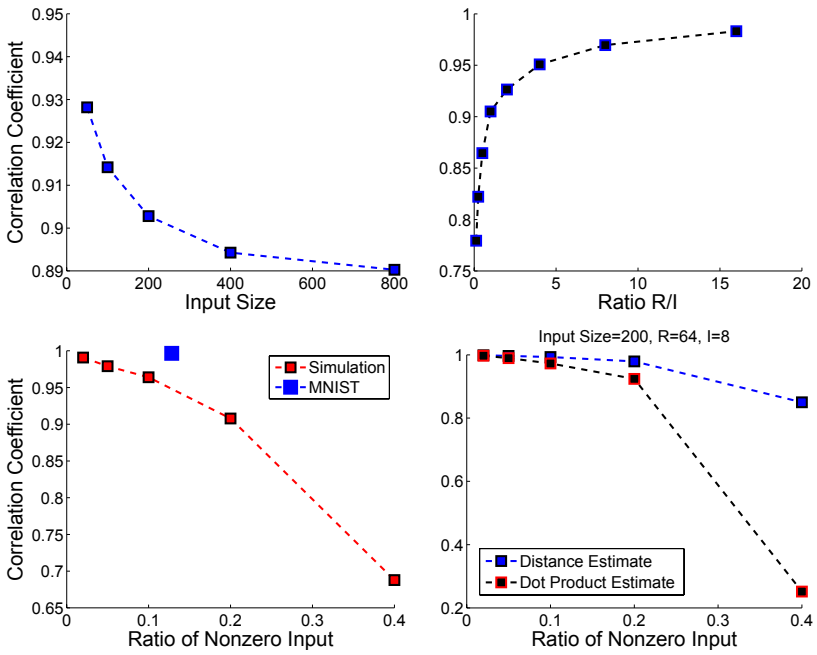

FIGURE 7
Results on metric learning simulations. The distance between data instances are estimated via linear regression using a set of relevant features. Average correlation coefficient is presented with respect to a set of factors: input size N, ratio of number of permutations and CA time evolutions R/I, ratio of non-zero elements in the input array Nz. The results are average of a set of factor combinations. Lower right plot shows the result for a specific configuration for both distance estimation and dot product estimation (see section 6).

us summarize the features for regression:

$$\mathbf{e_1} = {d_0}^T \; d_0$$
$$\mathbf{e_2} = {d_0}^T \; \mathbf{M_K} d_0$$
$$\mathbf{e_3} = {d_0}^T \; \mathbf{S_K} d_0$$

We ran a Monte Carlo simulation[5] to measure the quality of CA feature distance estimation according to a set of factors: size of data vector (N), ratio of nonzero elements (Nz), number of random permutations (R), number of CA evolutions (I). The distance features $[\mathbf{e_1}; \mathbf{e_2}; \mathbf{e_3}]$ are used for linear regression[6] of the CA feature distances of two data instances. A hundred random binary vectors are created, and their pairwise CA feature distances as

[5] 20 simulations per parameter combination.
[6] Linear regression was performed with pseudo inverse on the data matrix.

well as the distance features are computed. 80% of the CA feature distances (3960 pairs) are used for training the regressor, the rest of the pairs (990) are used for the test. The results are given in Figure 7. The mean correlation between the real distance and the estimate is:

1. moderately decreasing with vector size, leveling around 400 size inputs,
2. sharply increasing with R/I ratio, leveling around $R = 5I$,
3. linearly decreasing with ratio of nonzero elements.

The ratio R/I and ratio of nonzero elements determine the estimation quality, and we propose that it is due to the increased Kolmogorov complexity (uncompressibility) of the state evolutions in time [70].

The experiments indicate that, given an input set, it is possible to very accurately estimate CA feature distance metric with proper choice of parameters and preprocessing (subsampling explained in the next section). The data used in the simulations did not have any structure, yet it is possible to learn distances, which suggests that the major inherent factor in the learned metric is the ratio of nonzero elements (Nz) and vector size (N). We also tested the metric on MNIST handwritten digit recognition dataset (Figure 7, lower left). The first 200 digits were used to evaluate the metric learning performance, in which there is a large variance (mean 0.13 and std 0.04) on the ratio of nonzero elements[7]. The correlation coefficient is very high despite the variance in Nz, which suggests that the learned metric is robust also for structured data.

6 LINEAR COMPLEXITY KERNEL FOR SUPPORT VECTOR MACHINE LEARNING

We want to derive the kernel function for dot product/projection of CA features, that is intended to be used in support vector machines. The form of the proposed kernel derived through dot product in CA feature space enables the exchange of summation trick used in support vector machine linear kernels, that reduces computation time from $\mathcal{O}(DN)$ (D is the number of support vectors) to $\mathcal{O}(N)$. Given the fact that interesting problems require thousands of support vectors [14], this corresponds to a **threefold speedup**.

The projection of one CA feature over another is:

$$(\mathbf{A^R})^T . \mathbf{B^R} = {A_0}^T \, \mathbf{M}^T \; mod \; 2 \; . \; \mathbf{M} \; B_0 \; mod \; 2 \; .$$

[7] $R = 64$ and $I = 8$

Using the property of modulus and the steps similar as before we get,

$$(\mathbf{A^R})^T . \mathbf{B^R} \; mod \; 2 \; = {A_0}^T \; \mathbf{M_K} \; B_0 \; mod \; 2 \; .$$

It is possible to estimate the projection via learning a function discussed in the previous section:

$$(\mathbf{A^R})^T . \mathbf{B^R} = {A_0}^T \; \mathbf{M_K} \; B_0 + \; 2 \times \mathbf{g}({A_0}^T . \; B_0, \mathbf{M_K})$$
$$(\mathbf{A^R})^T . \mathbf{B^R} = \mathbf{f}({A_0}^T . \; B_0, \mathbf{M_K}).$$

The initial projection ${A_0}^T \; B_0$, weighted initial projection ${A_0}^T \; \mathbf{S_K} \; B_0$ and 'Mahalanobis' projection ${A_0}^T \; \mathbf{M_K} \; B_0$ can be used as the features (similar with $e1$, $e2$, $e3$ above) for learning the estimation function of the CA feature projection. Weighted initial projection is nothing more than a vector product of the form, $({A_0}^T \; .* \; \mathbf{s}) \; B_0$, in which $.*$ is elementwise product and $\mathbf{s}$ is the diagonal vector of $\mathbf{S_K}$. The quadratic term requires $\mathcal{O}(N^2)$ time due to matrix vector multiplication, but it will be performed offline only once during training, which is explained below.

Metric learning with linear regression gives three coefficients, and the projection estimate is given by,

$$(\mathbf{A^R})^T . \mathbf{B^R} = k_1 . \; {A_0}^T . \; B_0 + k_2 . \; ({A_0}^T \; .* \; \mathbf{s}) \; B_0 + k_3 . \; {A_0}^T \; \mathbf{M_K} \; B_0 .$$

The performance of the three metrics on estimation of CA feature similarity is given in Figure 7, lower right plot. It is observed that dot product estimate is as good as distance estimate, but breaks down for dense vectors (large ratio of nonzero elements). However if the data is dense, the detrimental effect on dot product estimation can always be avoided by subsampling during the random permutation initialization stage of the algorithm.[8]

Then in support vector machine (SVM) framework, we can use the newly defined kernel. The prediction function of an SVM is given by,

$$w^T . \; X^R = \sum_{i=1}^{D} \alpha_i \; y^{(i)} k(X, \mathbf{Y}^{(i)}),$$

where w is the weight, X^R is the input representation in CA feature space. This costly multiplication due to high dimensionality is replaced using the summation in the right hand side. D is the number of support vectors and $\mathbf{Y}$ is the matrix that holds them, $y^{(i)}$ is the output value (class label or regressed

[8]This will increase the computational demands, i.e. larger R and I will be neccessary. However, this is not an issue for the kernel method, because kernel matrix $\mathbf{M_K}$ will be computed once, offline.

variable) of the support vector, α_i is the weight, k is the kernel function, and X is the input raw data. Using the kernel obtained via linear regression over the defined features, we get,

$$w^T . X^R = \sum_{i=1}^{D} \alpha_i \, y^{(i)} \sum_{j=1}^{N} (k_1 . X_j \mathbf{Y}_j^{(i)} + k_2 . X_j \, \mathbf{U}_j^{(i)} + k_3 \, X_j \mathbf{Z}_j^{(i)})$$

where $\mathbf{U}^{(i)} = \mathbf{s} . * \mathbf{Y}^{(i)}$ and $\mathbf{Z}^{(i)} = \mathbf{M_K} \, \mathbf{Y}^{(i)}$.

Exchanging the place of summation terms and rearranging,

$$w^T . X^R = \sum_{j=1}^{N} X_j \sum_{i=1}^{D} \alpha_i \, y^{(i)} (k_1 \, \mathbf{Y}_j^{(i)} + k_2 . \mathbf{U}_j^{(i)} + k_3 \, \mathbf{Z}_j^{(i)}).$$

The second summation (denoted as Q_j) can be computed offline and saved, to be used during both training and test:

$$w^T . X^R = \sum_{j=1}^{N} X_j . Q_j .$$

There are three remarks about the kernelization of CA feature for linear rules:

1. Two new types of support vectors are defined $\mathbf{U}^{(i)}$ and $\mathbf{Z}^{(i)}$, and kernel is a weighted combination of the three support vectors, which resembles [40].
2. Standard echo state networks are kernelized in [31, 57]. These studies are enlightening because they bridge the gap between kernel based machine learning and recurrent neural networks. Yet the computational complexity of the proposed recursive kernelizations in these studies is at least as large as nonlinear kernel methods.
3. There are studies for imitating nonlinear kernels via polynomial approximations [14, 20]. They transform the computational complexity from $\mathcal{O}(DN)$ to $\mathcal{O}(N^2)$, which gives speedups only for low dimensional data.

However, different from the previous studies, **the computational complexity of the cellular automata kernel is identical to standard linear kernel, $\mathcal{O}(N)$.** Yet, linear cellular automata rules are known to be very powerful computational tools, comparable to Turing complete rules (see Tables 3 and 5, [8, 10, 50, 70]).

Dataset	Linear Raw	CA Kernel	RBF Kernel	Polynomial Kernel
MNIST	85	86	86.5	82.5

TABLE 5
Performance comparison of classifiers on subset of MNIST dataset. See text for details.

How does the CA kernel perform on structured data? We compared[9] it with other SVM kernels [16] on MNIST dataset [44]. We used the first 200 data instances (binarized with intensity threshold 128) to learn the kernel function (two features). We used the kernel function to get the kernel matrix (pairwise projections between all data instances), then trained a classifier using the estimated kernel matrix. We tested the classifier on the next 200 data points again using the estimated kernel matrix. The results are given in Table 5 where we compare linear , CA ($R = 128$ and $I = 8$), RBF and polynomial (order 2) kernels. Coarse-to-fine grid search is done on parameter space for finding the best performance for all classifiers. It is observed that CA kernel approaches RBF kernel, even though the computational complexity is linear. Interestingly, polynomial kernel severely overfits for orders higher than 2, yet the CA kernel estimates 8^{th} order statistics (because $I = 8$) but does not overfit which shows a clear advantage of distributed representation (see also Table 4). Having said that, the true power of CA computation is tapped only when hybrid/nonlinear rule based, multi-layered CA feature spaces are built that utilize unsupervised pre-training (see Discussion section for potential improvements).

7 ANALYSIS OF THE FEEFORWARD CELLULAR AUTOMATA NETWORK

The cellular automaton feature space can be perceived as a network, an illustration for rule 90 is given in Figure 8 ($N = 5$). In one view (**a** in the figure), the network consists of $R \times I$ number of connected components shown in a grid. In this representation, the time dimension in CA evolution is **unrolled** and given in the columns, where the adjacency matrix for time k is determined by $M_N^k \ mod \ 2$[10]. Each permutation is a copy of the time unrolled network, that receives from a different permutation of inputs, given as rows. For

[9]The experiments were not performed using the whole datasets or large (R, I) combinations due to memory issues. Memory efficient implementation is necessary because off-the-shelf libraries are not fit for our framework. A comparison with state-of-the-art is planned for the near future.

[10]Modulus operation simplifies M_N because even entries are ineffective and they are zeroed out after the modulus.

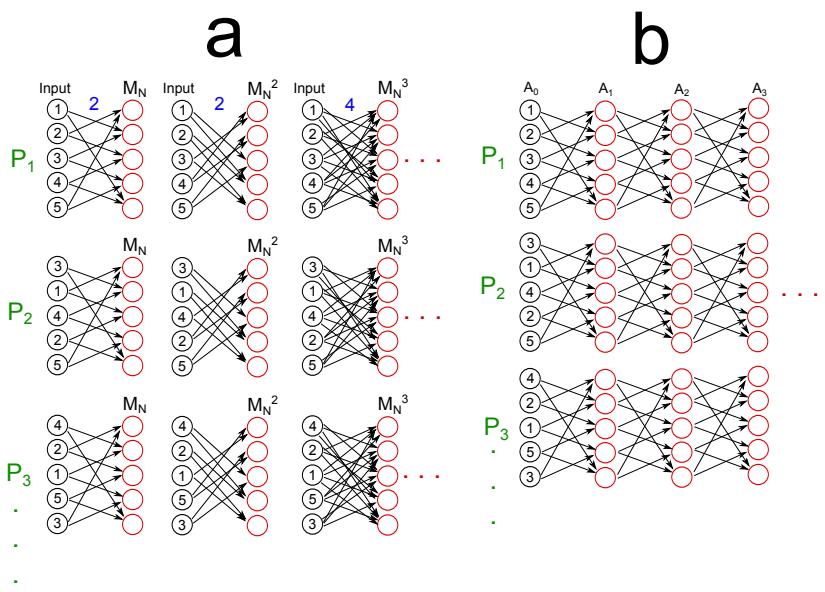

FIGURE 8
The network formed by the CA feature space of rule 90 CA. **a.** The network can be viewed as a collection of connected components, which is composed of different input permutations (P, in rows) and time unrolled CA evolutions (in columns). The average number of connections per unit increases with time, as shown in blue in the uppermost row. **b.** The network can be viewed as a time unrolled feedforward network, however the connections are not all-to-all between layers due to partitioning of different permutations, given as separate rows.

simplicity of illustration, the input projection (the indices 1,2,3,4,5 are shown for input vector) is demonstrated separately for each connected component. The average number of connections increases with the time evolution (see blue numbers below the connected components in the first row), because $M_N^k \ mod \ 2$ density increases with k. This observation is compatible with the Kolmogorov complexity experiments in [70], that state the increase in uncompressibility of CA states in time. One other perspective on the uncompressibility is related with the pattern replication property of rule 90 [66]. As time evolves, it is less likely to replicate a portion of the initial pattern, due to increased number of interactions (i.e. more connections with increasing I in the network, in Figure 8), hence reduce the probability of effective data compression.

There are $N \times I \times R$ neurons in this feedforward network. The neurons can be arranged to obtain the a recurrent neural network equation in classical form. Suppose A_0^R is a $N \times R$ size matrix which holds the initial conditions

(i.e. raw data) of the CA for R different permutations. Then,

$$A_1^R = M_N A_0^R \ mod\ 2$$
$$A_2^R = M_N A_1^R \ mod\ 2$$
$$...$$
$$A_k^R = M_N A_{k-1}^R \ mod\ 2,$$

where M_N is a the characteristic matrix defining state evolution of the linear CA defined in section 5. The recursive computation definition of recurrent neural network is in autonomous mode (i.e. zero input), and this is due to flattening of the input. Using this recurrent formulation, CA expansion can also be viewed as a time unrolled continuous (i.e. not connected components) network shown in Figure 8 b.

How about nonlinear CA rules? We can write a generalized the CA network equation as follows:

$$A_k^R = \mathbf{f}(A_{k-1}^R),$$

where function $\mathbf{f}$ is over the neighbors of the cells and it approximates the rule of the CA. If the rule is not 'linearly separable' then we have to use multiple layers. Therefore, each time step in Figure 8 b might require a number of hidden layers. The overall network is a repetition (at each time step) of a multi-layer perceptron, with heaviside step nonlinearity at the last layer. To the best of our knowledge, this is a novel perspective, and 'neuralization' of the CA rules seems like an interesting research direction.

8 CELLULAR AUTOMATA RESERVOIR

In feedforward CA feature expansion experiments we presented the sequence as a whole by flattening: whole input sequence is vectorized, CA feature is computed, then whole output feature is estimated using this large holistic sequence representation. Using a larger data portion for context is common in sequence learning and it is called sliding window method, and estimating the whole output sequence is generally utilized in hidden Markov model approaches [23], but these two were not applied together before to the best of our knowledge. We adopted this approach for 5/20 bit memorization tasks, however it is possible to estimate each time step in the output sequence one at a time, as in echo state networks. Suppose that we would like to estimate O_t, the output at time step t. Using Markovian assumption we say that only the input up to M steps back is relevant, then we use the chunk of the input sequence $(I_{t-M}; I_{t-M-1}; ...I_{t-1})$ as an input to the cellular automata reservoir.

This assumption is widely used in classical recurrent neural network studies, in which the hidden layer activities are reset at every M time steps (eg. see Language Modeling in [55]). We use this chunk of input as initial conditions of the cellular automata, compute CA evolution and use CA reservoir feature vector for regression/classification. Why didn't we use this 'classical' approach in 5 bit and 20 bit tasks in section 3? We have to discard the first M output time steps because we need M input time steps to estimate the $M+1^{st}$ time step of the output sequence. However, if we had discarded the first M time steps, it wouldn't be a fair comparison with results on echo state networks. Please note that, estimating the whole sequence using the whole input sequence is a much harder task, and we are making our case more difficult in order to keep the comparison with literature.

Let us emphasize that using the Markovian approach, CA reservoir is able to perform any task that echo state network (or recurrent neural network in general) is able to do. Yet, we have to choose the right amount of history needed for estimation (i.e. parameter M). The size of the representation increases with M in our algorithm. The advantage of echo state networks over our approach is its capability to keep a fixed size representation (activity of neurons in the reservoir) of timeseries history of arbitrary size. However, as the size of the timeseries history needed to perform the task increases, the size of the reservoir network should increase proportionally, suggested by the experiments in [37]. Then for feedforward cellular automata expansion there is a selection of history M that constitutes the feature space, and for echo state networks the number of neurons that accommodate timeseries history of M should be decided.

Feedforward cellular automata is shown to be capable of long-short-term-memory. Nevertheless we would like formulate a classical recurrent architecture using cellular automata feature expansion, because it is a much intuitive representation for sequential tasks.

Suppose that cellular automaton is initialized with the first time step input of the sequence, X_0. The vector ${X_0}^{P_1}$ is a single random permutation of the input. In contrast to feedforward formulation, we concatenate multiple random permutations to form a $N \times R$ vector in the beginning of CA evolution:

$$ {X_0}^P = [{X_0}^{P_1}; {X_0}^{P_2}; {X_0}^{P_3}; ... {X_0}^{P_R}] $$

Cellular automaton is initialized with ${X_0}^P$ and evolution is computed using a prespecified rule, Z (Figure 5):

$$ \begin{aligned} A_1 &= Z({X_0}^P), \\ A_2 &= Z(A_1), \\ &... \\ A_I &= Z(A_{I-1}). \end{aligned} $$

A_1 up to A_I are state vectors of the cellular automaton. We concatenate the evolution of cellular automata to obtain a single state vector of the reservoir (size NIR), to be used for estimation at time 1:

$$A^{(1)} = [A_1; A_2; ...A_I].$$

We have to insert the next time step of the input, X_1 into the reservoir state vector. There are many options, but here we adopt normalized addition of state vector and input vector [41], in which, entries with value 2 (i.e. $1+1$) become 1, with value 0 stay 0 (i.e. $0+0$), and with value 1 (i.e. $0+1$) are decided randomly. We modify the state vector of the cellular automaton at time I:

$$A_I = [A_I + {X_1}^P],$$

in which square brackets represent normalized summation. The cellular automaton is evolved for I steps to attain $A^{(2)}$,

$$A^{(2)} = [A_{I+1}; A_{I+2}; ...A_{2I}],$$

which is used for estimation at time step 2. This procedure is continued until the end of the sequence, when X_T is inserted into the reservoir.

We analyzed the recurrent evolution for additive rule 90^{11}, as we did in section 4. Interestingly, the recurrence at multiples of I^{th} time step is the same with heteroassociative storage of sequences proposed in [26, 41, 56]:

$$A_I = \Pi_I {X_0}^P \oplus \Pi_I {X_0}^P,$$

$$A_{2I} = \Pi_I({X_1}^P \oplus \Pi_I {X_0}^P) \oplus \Pi_{-I}({X_1}^P \oplus \Pi_{-I} {X_0}^P)$$

$$A_{3I} = \Pi_I({X_2}^P \oplus \Pi_I({X_1}^P \oplus \Pi_I {X_0}^P)) \oplus \Pi_{-I}({X_2}^P \oplus \Pi_{-I}({X_1}^P \oplus \Pi_{-I} {X_0}^P))$$

...

However, different from [26, 41, 56], we are using the whole set of cellular automaton state vectors (i.e. I time steps of evolution) as the reservoir, and it can used for estimating the sequence output via any machine learning algorithm (SVM, logistic regression, multilayer perceptron etc.) other than autoassociative retrieval. Nevertheless, additive cellular automata is very similar to a linear recurrent architecture proposed by Kanerva, Gallant and previously by Plate. Yet, there is a large arsenal for cellular automata rules, most of

[11] For simplification I is assumed to be a power of 2.

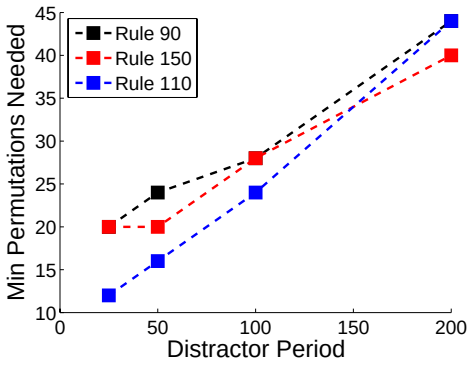

FIGURE 9
Distractor period vs the minimum number of permutations needed to obtain zero error in 5 bit memorization task is given for three different elementary cellular automata rules.

which are nonlinear and some of them Turing complete. Then our approach is a generalization of previously proposed recurrent architectures, in terms of atomic operation (i.e. linear vs nonlinear), readout (nearest neighbor retrieval vs a wide set of machine learning algorithms) and most importantly the representation (random vector/matrix vs cellular automata states).

We tested the performance of the cellular automata reservoir on 5 bit task. We used $I = 32$, and varied the number of permutations R, to obtain the minimum R value (y axis in Figure 9) to get zero error in 5 bit tasks of various distractor periods (x axis in Figure 9). In Figure 9, we are showing the results for three cellular automata rules. It is observed that reservoir size (i.e. $R \times I$) demand has increased compared to feedforward cellular automata feature expansion (Table 3). Yet memory requirements in training stage is much smaller, due to the abandonment of flattening. Most importantly, recurrent architecture enables generation of unbounded-length signal output, and acceptance of unbounded-length timeseries input.

Can we generalize our analyses on feedforward CA feature expansion for CA reservoir? Cellular automaton evolution and feature computation after insertion of a sequence time step ($A^{(1)}$ up to $A^{(T)}$) is equivalent to feedforward computation given in section 2. Then we can say that:

1. The cellular automaton states hold a distributed representation of attribute statistics, as well as correlation between sequence time steps (section 4).
2. The reservoir feature vector can be kernelized for efficient implementation in support vector machine framework (section 6).

3. The CA reservoir can be viewed as a time unrolled recurrent network with xor connections instead of algebraic synapses (section).

9 COMPUTATIONAL POWER

The computational power of a system has many aspects, and it is a matter under debate for both cellular automata [50] and reservoir computing [46]. The experiments on 5 bit and 20 bit noiseless memory tasks (Table 1) suggest that the capability increases with both number of permutations R and number of CA evolutions I. We also wanted to test other effects such as size of data vector (N), ratio of nonzero elements (Nz) as in section 5. In a simulation for rule 90, we computed mean pairwise correlation between CA features of randomly created vectors. The smaller this is, the larger the expected computational power [46]. The results are shown in Figure 10, which indicate that computational power increases with vector size and ratio of nonzero elements. We will extend these simulations for all elementary rules (including ECAM) and other metrics in a future work.

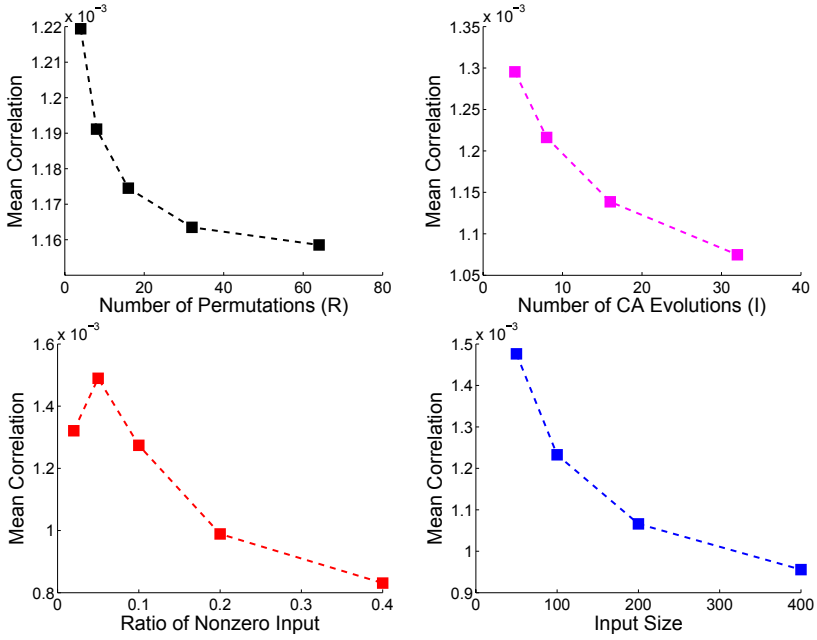

FIGURE 10
The simulation results that measures the mean pairwise correlation of the CA features for different inputs. A smaller number is an indicator of more computational power.

Task	ESN (Floating Point)	CA (Bitwise)	Speedup
5 bit $T_0 = 200$	1.03 M	0.43 M	2.4X
5 bit $T_0 = 1000$	13.1 M	8.3 M	1.6X
20 bit $T_0 = 200$	17.3 M	9.5 M	1.8X

TABLE 6
The comparison of the number of operations for the echo state networks (ESN) and the Cellular Automata (CA) framework. Operation is floating point for ESN, but bitwise for CA.

10 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 multiplications.
2. Since the CA feature vector is binary, matrix multiplication needed in the linear classification/regression can be replaced by summation.

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

The number of operations needed for the CA feature computation of 5 bit and 20 bit tasks is given in Table 6, both for Echo State Network (ESN) in [37] and for cellular automata (only feedforward architecture shown in the table).[12] There is a speedup in the number of operations in the order of 1.5-3X. However, considering the difference of complexity between floating point and bitwise operations, there is **almost two orders** of magnitude speedup/energy savings.[13] When looked into the CA reservoir, it is observed that about 3-4 times more computation is needed compared to feedforward feature expansion, yet it is still at least an order of magnitude faster than neuron based approaches.

When support vector machines are used for linear cellular automaton rule kernels, the computational complexity is $\mathcal{O}(N)$, in which N is the size of the initial data vector. However computational complexity is $\mathcal{O}(NIR)$ for

[12] For ESN, it is assumed that the number of floating point operations is equal to 2*NNZ.

[13] CPU architectures are optimized for arithmetic operations: bitwise logic takes 1 cycle and 32 bit floating point multiplication takes only 4 cycles, on 4th generation Haswell Intel^{TM} core. Therefore the speedup/energy savings due to the bitwise operations will be much more visible on hardware design, i.e. FPGA.

general rules, where R is the number of permutations and I is the number of CA evolutions[14]. Given the fact that interesting problems would require $R > 30$ and $I > 30$ (eg. Table 2), then usage of linear CA rule kernels will provide at least **three orders of magnitude** speedups[15]. The same argument is valid when we compare CA kernel with nonlinear kernels (eg. RBF), where the complexity is $\mathcal{O}(DN)$, where D is the number of support vectors in the order of thousands for interesting problems.

11 DISCUSSION

We provide novel feedforward and reservoir computing frameworks that are capable of long-short-term-memory, which require significantly less computation compared to echo state networks (Table 6).[16] In the proposed 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 automata space. Several theoretical advantages of the cellular automata framework compared to echo state networks are mentioned, in addition to their practical benefits. Cellular automata are easier to analyze and have insurances on Turing completeness. From CA side of the medallion: the computation performed in cellular automata can be conceptualized in many levels [38,53,67], but **our main proposition is that, reservoir computing is a very good fit for harnessing the potential of cellular automata computation.**

11.1 Distributed Representation on CA states

Some of the best performing cellular automata rules are additive. How does extremely local, additive and bitwise computation gives surprisingly good performance in a pathological machine learning task? This question needs further examination, however the experiments (see Section 3) suggest that if a dynamical system has universal computation capability, it can be utilized 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 enhanced the probability of long range interactions (Table 3).

In the paper we prove that cellular automata feature space holds a distributed representation of higher order attribute statistics, and it is more

[14] Because linear methods are applied on CA feature space of size NIR.

[15] For 20 bit task, $R \times I \approx 6000$.

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

effective than local representation (Table 4). Second order statistics (covariance matrix) is shown to be very informative for a set of AI tasks [60], yet we are providing a novel way of exploiting higher order statistics using cellular automata.

The usage of cellular automata for memorizing attribute statistics is in line with Elementary Callular Automata with Memory (ECAM). ECAM uses history by including previous states in every cell iteration. Reservoir computing based CA proposed in this paper uses classical CA rules, do not crash current cell states with previous states, but record all the history in a large reservoir. [8, 51] show that the memory function can push a CA rule into a completely different computation regime and this can be used in our framework for rule hybridization process in the reservoir. Also ECAM will provide a neat way of handling sequences without the need for flattening.

11.2 Cellular Automata Kernel Machines

As it is shown in [61] that the distance metrics in covariance statistics can be learned, we provide the theoretical foundation for distance metric learning of CA features for linear cellular automata rules (section 5). We show that the distance estimates are very accurate for a wide range of parameter settings (Figure 7). More importantly, linear cellular automata features can be kernelized to be used in support vector machine framework (section 6), and they have the same computational complexity with linear kernel while approaching the performance of nonlinear kernels (Table 5). Linear CA rules are shown to be powerful computational tools [8, 10, 50, 70], yet we can use very low complexity kernel methods to estimate their feature space. The kernelization effort is parallel to recurrent kernel machines framework [31, 57] and further experiments are needed to quantify the performance of CA kernelization and comparison with recurrent kernel machines.

11.3 Extensions and Implementation

There are a few extensions of the framework that is expected to improve the performance and computation time:

1. A hybrid [59] and a multilayer automaton can be used to handle different spatio-temporal scales in the input. Two types of hybrid, in one is CA space is hybrid, in another after each random permutation a different rule is applied.
2. The best rule/random mapping combination can be searched in an unsupervised manner (pre-training). The rank of the binary state space can be used as the performance of combinations. Genetic algorithm based search is also a good option for non-elementary rules [22].

3. We can devise ECAM rules in the reservoir instead of classical rules. In that approach we will have another way of achieving long range attribute interactions, and hybridization.

Cellular automata are very easy to implement in parallel hardware such as FPGA ([30]) or GPU (unpublished experiments on [1]). 100 billion cell operations per second seem feasible on mid-range GPU cards, this is a very large number considering 10 million operations are needed for 20 bit task.

11.4 Recurrent Computation in Cellular Automata Reservoir

We provide a recurrent architecture (section 8), that holds a fixed sized representation (cellular automata state vector) for a sequence of arbitrary length. The proposed algorithm is able to generate data as in echo state networks. This capability needs to be tested, possibly using language task such as character and word prediction. The results on noiseless memorization task is encouraging, yet there are many possibilities for improvement, some of them mentioned above.

11.5 Potential Problems

There is one superficial problem with the proposed framework: CA feature expansion is expected to vastly increase the feature dimension for complicated tasks (R and I are both very large), and curse with dimensionality. However, linear kernel that will be used in large CA feature space is known to be very robust against large dimensionality (deduced using structural risk minimization theory, [64]). Although linear kernel behaves nicely, the remaining problem due to the large dimensionality of the feature space can be alleviated by using a bagging approach that also selects a subset of the feature space in each bag [43] or the kernel method proposed in section 6 for linear CA rules.

As a future work we would like to test the framework on large datasets for language modeling, music/handwriting prediction and computer vision.

ACKNOWLEDGMENTS

This research is supported by The Scientific and Technological Research Council of Turkey (TUBİTAK) Career Grant, No: 114E554 and Turgut Ozal University BAP Grant, No: 006-10-2013. I would like to thank Lenovo Group Ltd. Turkey Division, specifically Cagdas Ekinci and Alpay Erturkmen for their support in this research.

A DETAILS OF EXPERIMENTS

Here we provide the details of the experiments performed in the paper. The Matlab® codes for all experiments are shared in http://www.ozguryilmazresearch.netozguryilmazresearch.net. The appendix is partitioned into sections the same name as the ones in the main body of the paper.

A.1 Cellular Automata Feature Space

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 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 Memory of Cellular Automata State Space

In these experiments, the Matlab® code kindly provided by [37] is used without much change. The echo state network based reservoir computation is replaced by cellular automata feature expansion. The sequence is flattened by concatenating the time steps into a large binary vector. Then the CA feature space receives it as the initial conditions, and computes the evolution for many different random permutations. Inclusion of the original data vector to the CA feature vector does not alter the performance significantly (interestingly exclusion performs better), and it is included. Increasing the distractor period expands the size of the vector, and adds irrelevant data to the CA computation reducing the estimation quality. Pseudoinverse based regression is used to estimate the output vector, all at once. 300 data points are used for training in 20 bit task, and only 10 time steps are used from the distractor period (called NpickTrain) during training. Yet, regression is expected to give the vector for full distractor period. 100 test data are used for test. 25 trials are averaged.

A.3 Distributed Representation of Higher Order Statistics

In this experiment we used the 5 bit memory task to compare the distributed and local representation of attribute statistics. For doing that, we computed the C_k vectors instead of A_k (CA evolution at k_{th} time step) for each time step and each permutation and formed an identical sized feature vector:

$$C_1 = A_1$$

$$C_2 = A_2$$

$$C_3 = A_3 \oplus A_1$$

$$C_4 = A_4$$

$$C_5 = A_5 \oplus A_3 \oplus A_1$$

$$C_6 = A_6 \oplus A_1$$

$$C_7 = A_7 \oplus A_5 \oplus A_1$$

$$C_8 = A_8$$

$$FeatureVector = [C_1; C_2; C_3; C_4; C_5; C_6; C_7; C_1].$$

Then we used that vector for regression.

A.4 Metric Learning for Linear Cellular Automata Feature Space

The information given in the main body is enough to replicate the simulations.

A.5 Linear Complexity Kernel for Support Vector Machine Learning

The information given in the main body is enough to replicate the experiment performed on MNIST dataset.

A.6 Computational Power

The information given in the main body is enough to replicate the simulations.

A.7 Cellular Automata Reservoir

In these experiments, again the Matlab® code kindly provided by [37] is used without much change. The echo state network based reservoir computation is replaced by cellular automata reservoir. The details of the reservoir features are given in section 8. Pseudoinverse based regression is used to estimate each sequence output, one at a time. 20 trials are averaged.

REFERENCES

[1] Conway's game of life on gpu using cuda. http://www.marekfiser.com/Projects/Conways-Game-of-Life-on-GPU-using-CUDA. Published: March 2013.

[2] Andrew Adamatzky. *Game of life cellular automata*. Springer.

[3] Andrew Adamatzky. (2001). *Computing in nonlinear media and automata collectives*. CRC Press.

[4] Andrew Adamatzky, Benjamin De Lacy Costello, and Tetsuya Asai. (2005). *Reaction-diffusion computers*. Elsevier.

[5] Andrew Adamatzky and Jérôme Durand-Lose. (2012). *Collision-based computing*. Springer.

[6] Andrew I Adamatzky. (1994). *Identification of cellular automata*. CRC Press.

[7] Charu C Aggarwal and Philip S Yu. (2000). *Finding generalized projected clusters in high dimensional spaces*, volume 29. ACM.

[8] Ramon Alonso-Sanz and Margarita Martin. (2006). Elementary cellular automata with elementary memory rules in cells: The case of linear rules. *J. Cellular Automata*, 1(1):71–87.

[9] Cédric Archambeau, Frédéric Vrins, Michel Verleysen, *et al*. (2004). Flexible and robust bayesian classification by finite mixture models. In *ESANN*, pages 75–80. Citeseer.

[10] Jan M Baetens and Bernard De Baets. (2010). Phenomenological study of irregular cellular automata based on lyapunov exponents and jacobians. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(3):033112.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

[12] Nils Bertschinger and Thomas Natschläger. (2004). Real-time computation at the edge of chaos in recurrent neural networks. *Neural computation*, 16(7):1413–1436.

[13] JB Butcher, David Verstraeten, Benjamin Schrauwen, CR Day, and PW Haycock. (2013). Reservoir computing and extreme learning machines for non-linear time-series data analysis. *Neural networks*, 38:76–89.

[14] Hui Cao, Takashi Naito, and Yoshiki Ninomiya. (2008). Approximate rbf kernel svm and its applications in pedestrian classification. In *The 1st International Workshop on Machine Learning for Vision-based Motion Analysis-MLVMA'08*.

[15] Marcin Chady and Riccardo Poli. (1998). *Evolution of cellular-automaton-based associative memories*. Springer.

[16] Chih-Chung Chang and Chih-Jen Lin. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[17] Parimal Pal Chaudhuri. (1997). *Additive cellular automata: theory and applications*, volume 1. John Wiley & Sons.

[18] Leon O Chua and Lin Yang. (1988). Cellular neural networks: Applications. *Circuits and Systems, IEEE Transactions on*, 35(10):1273–1290.

[19] Leon O Chua, Sook Yoon, and Radu Dogaru. (2002). A nonlinear dynamics perspective of wolfram's new kind of science part i: Threshold of complexity. *International Journal of Bifurcation and Chaos*, 12(12):2655–2766.

[20] Marc Claesen, Frank De Smet, Johan AK Suykens, and Bart De Moor. (2014). Fast prediction with svm models containing rbf kernels. *arXiv preprint arXiv:1403.0736*.

[21] Matthew Cook. (2004). Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40.

[22] Rajarshi Das, Melanie Mitchell, and James P Crutchfield. (1994). A genetic algorithm discovers particle-based computation in cellular automata. In *Parallel problem solving from naturePPSN III*, pages 344–353. Springer.

[23] Thomas G Dietterich. (2002). Machine learning for sequential data: A review. In *Structural, syntactic, and statistical pattern recognition*, pages 15–30. Springer.

[24] Kenji Doya. (1992). Bifurcations in the learning of recurrent neural networks 3. *learning (RTRL)*, 3:17.

[25] Ken-ichi Funahashi and Yuichi Nakamura. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806.

[26] Stephen I Gallant and T Wendy Okaywe. (2013). Representing objects, relations, and sequences. *Neural computation*, 25(8):2038–2078.

[27] Niloy Ganguly, Biplab K Sikdar, and P Pal Chaudhuri. (2001). Theory of additive cellular automata. *Fundamenta Informaticae XXI*, pages 1001–1021.

[28] Niloy Ganguly, Biplab K Sikdar, Andreas Deutsch, Geoffrey Canright, and P Pal Chaudhuri. (2003). A survey on cellular automata.

[29] Yoav Goldberg and Michael Elhadad. (2008). splitsvm: fast, space-efficient, non-heuristic, polynomial kernel computation for nlp applications. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 237–240. Association for Computational Linguistics.

[30] Mathias Halbach and Rolf Hoffmann. (2004). Implementing cellular automata in fpga logic. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 258. IEEE.

[31] Michiel Hermans and Benjamin Schrauwen. (2012). Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133.

[32] Sepp Hochreiter. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*.

[33] Sepp Hochreiter and Jürgen Schmidhuber. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[34] John J Hopfield. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

[35] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501.

[36] Herbert Jaeger. (2001). The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34.

[37] Herbert Jaeger. (2012). Long short-term memory in echo state networks: Details of a simulation study. Technical report, Technical Report.

[38] P James, Melanie Mitchell, and Rajarshi Dasz. The evolutionary design of collective computation in cellular automata.

[39] J Javier Martínez, Javier Garrigós, Javier Toledo, and J Manuel Ferrández. (2013). An efficient and expandable hardware implementation of multilayer cellular neural networks. *Neurocomputing*, 114:54–62.

[40] Tony Jebara. (2004). Multi-task feature and kernel selection for svms. In *Proceedings of the twenty-first international conference on Machine learning*, page 55. ACM.

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

[42] Brian Kulis. (2012). Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364.

[43] Patrice Latinne, Olivier Debeir, and Christine Decaestecker. (2000). Mixing bagging and multiple feature subsets to improve classification accuracy of decision tree combination. In *BENELEARN 2000, Tenth Belgian-Dutch Conference on Machine Learning*.

[44] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[45] Robert Legenstein and Wolfgang Maass. (2007). Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334.

[46] Mantas LukošEvičIus and Herbert Jaeger. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

[47] Wolfgang Maass, Thomas Natschläger, and Henry Markram. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

[48] Jianchang Mao and Anil K Jain. (1996). A self-organizing network for hyperellipsoidal clustering (hec). *Neural Networks, IEEE Transactions on*, 7(1):16–29.

[49] Olivier Martin, Andrew M Odlyzko, and Stephen Wolfram. (1984). Algebraic properties of cellular automata. *Communications in mathematical physics*, 93(2):219–258.

[50] Genaro J Martinez. (2013). A note on elementary cellular automata classification. *arXiv preprint arXiv:1306.5577*.

[51] Genaro J Martinez, Andrew Adamatzky, and Ramon Alonso-Sanz. (2013). Designing complex dynamics in cellular automata with memory. *International Journal of Bifurcation and Chaos*, 23(10).

[52] Jerry M Mendel. (1991). Tutorial on higher-order statistics (spectra) in signal processing and system theory: theoretical results and some applications. *Proceedings of the IEEE*, 79(3):278–305.

[53] Melanie Mitchell *et al.* (1996). Computation in cellular automata: A selected review. *Nonstandard Computation*, pages 95–140.

[54] Yuichi Motai. (2015). Kernel association for classification and prediction: A survey. *Transactions on Neural Networks and Learning Systems, IEEE*, 26(2).

[55] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. (2012). On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*.

[56] Tony A Plate. (2003). Holographic reduced representation: Distributed representation for cognitive structures.

[57] Zhiwei Shi and Min Han. (2007). Support vector echo-state machine for chaotic time-series prediction. *Neural Networks, IEEE Transactions on*, 18(2):359–372.

[58] Hava T Siegelmann and Eduardo D Sontag. (1995). On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150.

[59] Moshe Sipper, Marco Tomassini, and Mathieu S Capcarrere. (1998). Evolving asynchronous and scalable non-uniform cellular automata. In *Artificial Neural Nets and Genetic Algorithms*, pages 66–70. Springer.

[60] Oncel Tuzel, Fatih Porikli, and Peter Meer. (2006). Region covariance: A fast descriptor for detection and classification. In *Computer Vision–ECCV 2006*, pages 589–600. Springer.

[61] Oncel Tuzel, Fatih Porikli, and Peter Meer. (2008). Pedestrian detection via classification on riemannian manifolds. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1713–1727.

[62] Panagiotis G Tzionas, Philippos G Tsalides, and Adonios Thanailakis. (1994). A new, cellular automaton-based, nearest neighbor pattern classifier and its vlsi implementation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(3):343–353.

[63] Vladimir Vapnik. (2000). *The nature of statistical learning theory*. Springer Science & Business Media.

[64] Vladimir Naumovich Vapnik and Vlamimir Vapnik. (1998). *Statistical learning theory*, volume 1. Wiley New York.

[65] David Verstraeten, Benjamin Schrauwen, Michiel dHaene, and Dirk Stroobandt. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403.

[66] Stephen Wolfram. (1983). Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601.

[67] Stephen Wolfram. (2002). *A new kind of science*, volume 5. Wolfram media Champaign.

[68] Liu Yang and Rong Jin. (2006). Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2.

[69] Ozgur Yilmaz. (2015). Classification of occluded objects using fast recurrent processing. *arXiv preprint arXiv:1505.01350*.

[70] Hector Zenil and Elena Villarreal-Zapata. (2013). Asymptotic behaviour and ratios of complexity in cellular automata. *arXiv preprint arXiv:1304.2816*.