Reservoir Computing Using Nonuniform Binary Cellular Automata

Stefano Nichele
Department of Computer Science
Oslo and Akershus University College of Applied Sciences
Oslo, Norway
stefano.nichele@hioa.no

Magnus S. Gundersen
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
magnugun@stud.ntnu.no

The reservoir computing (RC) paradigm utilizes a dynamical system (a reservoir) and a linear classifier (a readout layer) to process data from sequential classification tasks. In this paper, the usage of cellular automata (CAs) as a reservoir is investigated. The use of CAs in RC has been showing promising results. In this paper, it is shown that some cellular automaton (CA) rules perform better than others and the reservoir performance is improved by increasing the size of the CA reservoir itself. In addition, the usage of parallel loosely coupled (nonuniform) CA reservoirs, where each reservoir has a different CA rule, is investigated. The experiments performed on nonuniform CA reservoirs provide valuable insights into CA reservoir design. The results herein show that some rules do not work well together, while other combinations work remarkably well. This suggests that nonuniform CAs could represent a powerful tool for novel CA reservoir implementations.

1. Introduction

Complex real-life problems often require processing of time series data. Systems that process such data must remember inputs from previous time steps in order to make correct predictions in a future time step; that is, they must have some sort of memory. Recurrent neural networks (RNNs) have been shown to possess such memory [1].

Unfortunately, training RNNs using traditional methods like gradient descent is difficult [2]. A fairly novel approach called reservoir computing (RC) has been proposed [3, 4] to mitigate this problem. RC splits the RNN into two parts: the untrained recurrent part (a reservoir) and the trainable feed-forward part (a readout layer).

In this paper, an RC system is investigated, where a cellular automaton (CA) [5] computational substrate is used as the reservoir.

https://doi.org/10.25088/ComplexSystems.26.3.225

This approach to RC was proposed in [6] and further studied in [7–9]. The term ReCA is used as an abbreviation for "reservoir computing using cellular automata," and is adopted from the latter paper.

In the work herein, a fully functional ReCA system is implemented and extended into a parallel nonuniform CA reservoir system (loosely coupled). Various configurations of parallel reservoirs are tested and compared to the results of a single-reservoir system. This approach is discussed, and insights into different configurations of CA reservoirs are given.

2. Background

2.1 Reservoir Computing

Feedforward neural networks (NNs) are neural network models without feedback connections; that is, they are not aware of their own outputs [1]. They have gained popularity because of their ability to be trained to solve classification tasks. Examples include image classification [10] or playing the board game Go [11]. However, when trying to solve problems that include sequential data, such as sentence analysis, they often fall short [1]. For example, sentences may have different lengths, and the important parts may be spatially separated even for sentences with equal semantics. Recurrent neural networks (RNNs) can overcome this problem [1], being able to process sequential data through memory of previous inputs that are remembered by the network. This is done by relieving the NN of the constraint of not having feedback connections. However, networks with recurrent connections are notoriously difficult to train by using traditional methods [2].

Reservoir computing (RC) is a paradigm in machine learning that combines the powerful dynamics of an RNN with the trainability of a feedforward NN. The first part of an RC system consists of an untrained RNN, called a reservoir. This reservoir is connected to a trained feedforward neural network, called a readout layer. This setup can be seen in Figure 1.

The field of RC has been proposed independently by two approaches, namely echo state networks (ESN) [3] and liquid state machines (LSM) [4]. By examining these approaches, important properties of reservoirs are outlined.

Perhaps the most important feature is the echo state property [3]. Previous inputs "echo" through the reservoir for a given number of time steps after the input has occurred and thereby slowly disappear without being amplified. This property is achieved in traditional RC approaches by clever reservoir design. In the case of ESN, this is

achieved by scaling of the connection weights of the recurrent nodes in the reservoir [12].

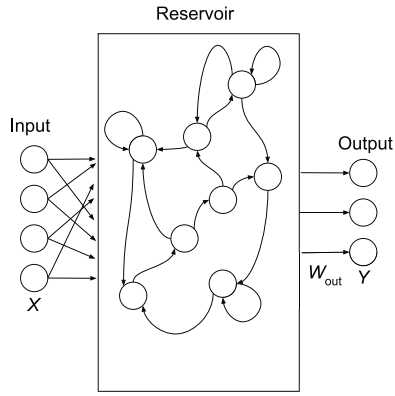

Figure 1. General RC framework. Input X is connected to some or all of the reservoir nodes. Output Y is usually fully connected to the reservoir nodes. Only the output weights W_{out} are trained.

As discussed in [13], the reservoir should preferably exhibit edge of chaos behaviors [14], in order to allow for high computational power [15].

2.2 Various Reservoir Computing Approaches

Different RC approaches use reservoir substrates that exhibit the desired properties. In [16] an actual bucket of water is implemented as a reservoir for speech recognition, and in [17] the *E. coli* bacteria is used as a reservoir. In [18] the primary visual cortex of an anesthetized cat was used as a reservoir, and in [19–21] unconventional carbon nanotube materials have been configured as a reservoir through artificial evolution. In [22, 23] an optoelectronic reservoir implementation is presented. In [24] and more recently in [25], the usage of random Boolean networks (RBNs) as reservoirs is explored. Random Boolean networks can be considered as an abstraction of CAs [26], and their use is thereby a related approach to the one presented in this paper.

2.3 Cellular Automata

A cellular automaton (CA) is a computational model, first proposed by Ulam and von Neumann in the 1940s [5]. It is a complex, decentralized and highly parallel system, in which computations may emerge [27] through local interactions and without any form of centralized control. Some CAs have been proved to be Turing complete

https://doi.org/10.25088/ComplexSystems.26.3.225

[28], that is, having all properties required for computation: transmission, storage and modification of information [14].

A CA usually consists of a grid of cells, each cell with a current state. The state of a cell is determined by the update function f, which is a function of the neighboring states n. This update function is applied to the CA for a given number of iterations. These neighbors are defined as a number of cells in the immediate vicinity of the cell itself.

In this paper, only one-dimensional elementary CAs are used. This means that the CA only consists of a one-dimensional vector of cells, named A, each cell with state $S \in \{0, 1\}$. In all the figures in this paper, $S = 0$ is shown as white, while $S = 1$ is shown as black. The cells have three neighbors: the cell to the left, itself and the cell to the right. A cell is a neighbor of itself by convention. The boundary conditions at each end of the one-dimensional vector are usually solved by wraparound, where the leftmost cell becomes a neighbor of the rightmost, and vice versa.

The update function f, hereafter denoted rule Z, works accordingly by taking three binary inputs and outputting one binary value. This results in $2^8 = 256$ different rules. An example of such a rule is shown in Figure 2, where rule 110 is depicted. The numbering of the rules follows the naming convention described by Wolfram [29], where the resulting binary string is converted to a base-10 number. The CA is usually updated in synchronous steps, where all the cells in the one-dimensional vector are updated at the same time. One update is called an iteration, and the total number of iterations is denoted by I.

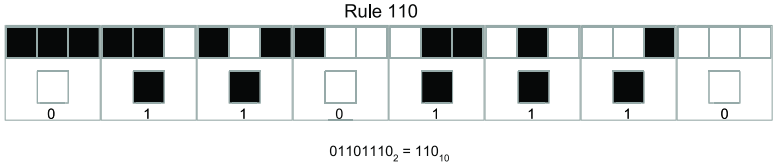

Figure 2. Elementary CA rule 110. The figure depicts all the possible combinations that the neighbors of a cell can have. A cell is its own neighbor by convention.

The rules may be divided into four qualitative classes [29] that exhibit different properties when evolved: class I evolves to a static state, class II evolves to a periodic structure, class III evolves to chaotic patterns, and class IV evolves to complex patterns. Class I and II rules will fall into an attractor after a short while [14] and behave in an orderly way. Class III rules are chaotic, which means that the

organization quickly descends into randomness. Class IV rules are the most interesting ones, as they reside at a phase transition between the chaotic and ordered phase, that is, at the edge of chaos [14]. In uniform CAs, all cells share the same rule, while nonuniform CA cells are governed by different rules. Quasiuniform CAs are nonuniform, with a small number of diverse rules.

2.4 Cellular Automata in Reservoir Computing

As proposed in [6], CAs may be used as a reservoir of dynamical systems. The conceptual overview is shown in Figure 3. Such a system is referred to as ReCA in [9], and the same name is therefore adopted in this paper. The projection of the input to the CA reservoir can be done in two different ways [6]. If the input is binary, the projection is straightforward, where each feature dimension of the input is mapped to a cell. If the input is nonbinary, the projection can be done by a weighted summation from the input to each cell. See [7] for more details.

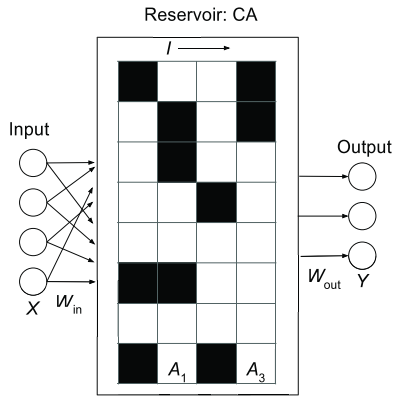

Figure 3. General ReCA framework. Input X is projected onto the cells of a one-dimensional CA, and the CA rule is applied for a number I of iterations. In the figure, each iteration is stored and denoted by A_i. The readout layer weights W_{out} are trained according to the target function. Figure adapted from [7].

The time evolution of the reservoir can be represented as follows:

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

where A_m is the state of the one-dimensional CA at iteration m and Z is the CA rule that was applied. A_0 is the initial state of the CA, often an external input, as discussed later.

As discussed in Section 2.1, a reservoir often operates at the edge of chaos [15]. Selecting CA-based reservoirs that exhibit this property is trivial, as rules that lie inside Wolfram class IV can provide this property. Additionally, to fully exploit such a property, all I iterations of the CA evolution are used for classification, and this can be stated as follows:

$$A = [A_1; A_2; \ldots A_I],$$

where A is used for classification.

The ReCA system must also exhibit the echo state property, as described in Section 2.1. This is done by allowing the CA to take external input, while still remembering the current state. As described in more detail later, ReCA systems address this issue by using a time-transition function, named F, which allows some previous inputs to echo through the CA.

Cellular automata also provide additional advantages to RC. In [7] a speedup of 1.5–3x in the number of operations compared to the ESN [30] approach is reported. This is mainly due to a CA relying on bitwise operations, while ESN uses floating-point operations. This can be additionally exploited by utilizing custom-made hardware like FPGAs. In addition, if edge-of-chaos rules are selected, Turing-complete computational power is present in the reservoir. Cellular automata theoretical analysis is easier than RNNs, and it allows Boolean logic and Galois field algebra.

2.5 Reservoir Computing Using Cellular Automata System Implementations

Reservoir computing using CA systems is a very novel concept, and therefore there are only a few implemented examples at the current stage of research. Yilmaz [6, 7] has implemented an ReCA system with uniform elementary CAs and Game of Life [31]. Bye [8] also demonstrated a functioning ReCA system. Kleyko et al. [32] used a uniform CA reservoir for image classification, and McDonald [33] proposed a CA reservoir as an extreme learning machine. Nichele and Molund [34] demonstrated that a CA reservoir can be used for deep learning. The approaches used are similar; however, there are some key differences, discussed in the following subsubsections.

2.5.1 Encoding and Random Mappings

In the encoding stage, [7] used random permutations over the same input vector. This encoding scheme can be seen in Figure 4. The permutation procedure is repeated R number of times, because it was

experimentally observed that multiple random mappings improve performance.

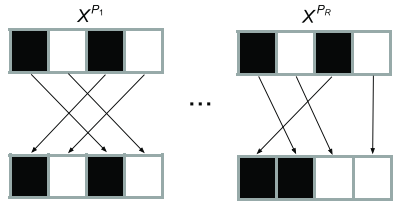

Figure 4. The encoding used in [7]. For a total of R permutations, X is randomly mapped to vectors of the same size as the input vector itself.

In [8] a similar approach was used. The main difference is that the input is mapped to a vector that is larger than the input vector itself. The size of this mapping vector is given by a parameter "automaton size." This approach can be seen in Figure 5. The input bits are randomly mapped to one of the bits in the mapping vector. The ones that do not have any mapping to them are left at zero.

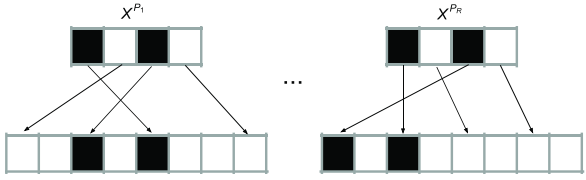

Figure 5. The encoding used in [8]. The input X is randomly mapped to a vector with size larger than the input vector itself. This mapping is done R times. The size of the vector that the input is mapped to can be determined in two ways. Either by "automaton size," which explicitly gives the size of the vector (in this case 8), or by the C parameter, where the size is given by $C * |X^{p_n}|$ (in this case $C = 2$).

In the work herein, the approach described in [8] is used, but with a modification. Instead of using the automaton size parameter, the C parameter is introduced. The total length of the permutation is given by the number C multiplied by the length of the input vector. In the case of Figure 5, the automaton size would be 8, and C would be 2.

2.5.2 Feedforward or Recurrent

Both a feedforward and a recurrent design were proposed in [7]. The difference was whether the whole input sequence is presented to the system in one chunk or step by step. Only a recurrent design was described in [8]. Only recurrent architectures will be investigated in this paper. This is because they are more in line with traditional

RNNs and RC systems and are conceptually more biologically plausible.

2.5.3 Concatenation of the Encoded Inputs before Propagating into the Reservoir

After random mappings have been created, there is another difference in the proposed approaches. In the recurrent architecture, [7] concatenates the R number of permutations into one large vector of length ($R * \text{input\_length}$) before propagating it in a reservoir of the same width as this vector. The one-dimensional input vector at time step t can be expressed as follows:

$$X_t^P = \left[X_t^{P_1}; X_t^{P_2}; X_t^{P_3}; \dots X_t^{P_R}\right].$$

X_t^P is inserted into the reservoir as described in Section 2.4 and then iterated I times. The iterations are then concatenated into the vector A^t, which is used for classification at time step t:

$$A^t = \left[A_1; A_2; \dots A_I\right].$$

A different approach was adapted in [8], the same one that was also used by the feedforward architecture in [7], where the R different permutations are iterated in separate reservoirs, and the different reservoirs are then concatenated before they are used by the classifier. The vector used for classification at time step t is as follows:

$$A^t = \left[A_{P_1}^t; A_{P_2}^t; \dots A_{P_R}^t\right],$$

where $A_{P_n}^t$ is the vector from the concatenated reservoir. In this paper, the recurrent architecture approach is used.

2.5.4 Time Transition

In order to allow the system to remember previous inputs, a time transition function is needed to translate between the current time step and the next. One possibility is to use normalized addition as the time transition function, as shown in Figure 6, with F as the normalized addition. This function works as follows: the cell values are added, and if the sum is 2 (1 + 1) the output value becomes 1; if the sum is 0, the output value becomes 0; if the sum is 1, the cell value is decided randomly (0 or 1). The initial one-dimensional CA vector of the reservoir at time step t is then expressed as:

$$A_0 = F\left(X_t, A_I^{t-1}\right), \quad t > 0,$$

where F may be any bitwise operation, X_t is the input from the sequential task at time step t, and A_I^{t-1} is the last iteration of the previous time step. At the first time step ($t = 0$), the transition function is bypassed, and the input X_t is used directly in the reservoir.

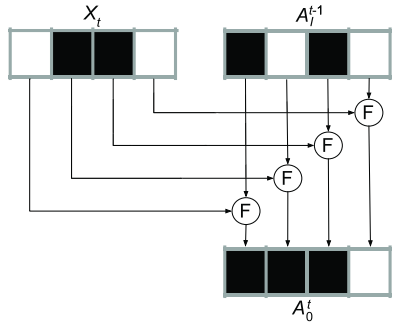

Figure 6. Time transition used in [7]. The sequence input X_t is combined with the state of the reservoir at the last iteration at the previous time step A_I^{t-1}. The function F may be any bitwise function. Only one permutation is shown in the figure to increase readability.

Another possibility is to use "permutation transition" as the time transition function, as seen in Figure 7. Here, all cells that have a mapping to them (from the encoder) are bitwise filled with the value of input vector X. If the cells do not have any mapping to them, the values from A_I^{t-1} are inserted. This allows the CA to have memory across time steps in sequential tasks. By adjusting the automaton size, or C parameter, the interaction between each time step can be regulated.

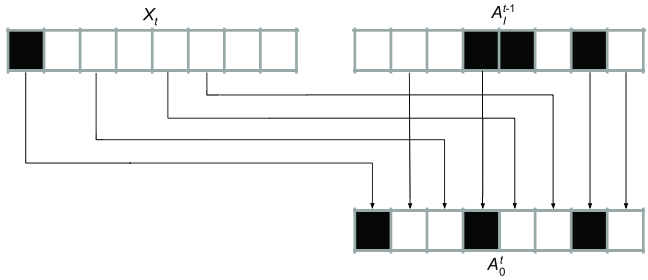

Figure 7. Time transition by permutation. The input is directly copied from X_t, according to the mapping from the encoder, as shown in Figure 5. The other cells have their values copied from the last iteration of the previous time step A_I^{t-1}. Only one permutation is shown to increase readability.

The described approaches have different effects on the parameters R and I and also the resulting size of the reservoir. This is relevant when discussing the computational complexity of ReCA systems.

In this paper, the "permutation transition" is used.

https://doi.org/10.25088/ComplexSystems.26.3.225

3. Experimental Setup

The basic architecture implemented in this paper is shown in Figure 8. The encoder is based on the architecture described in [8]. In this paper, the parameter C is introduced as a metric on how large the resulting mapping vector should be. The concatenation procedure is adapted from [7]. The vectors, after the encoding (random mappings), are concatenated into one large vector. This vector is then propagated into the reservoir, as described in Section 2.5.3. The time transition function is adapted from [8]. The mappings from the encoder are saved and used as a basis to which new inputs are mapped, as described in Section 2.5.4. The values from the last step in the previous time step are directly copied. The classifier used in this paper is a support vector machine, as implemented in the Python machine learning framework scikit-learn [35]. The code base that was used in this paper is available for download [36].

An example run with rule 90 is shown in Figure 9. This visualization gives valuable insights into how the reservoir behaves when parameters are changed and makes it easier to understand the reservoir dynamics. Most natural systems come in the form of a temporal system (sequential), that is, an input to the system depends on previous inputs. Classical feedforward architectures are known to have issues with temporal tasks [1]. In order to test the ReCA system at a temporal task, the five-bit task [37] is chosen in this paper. Such a task has become a popular and widely used benchmark for RC, in particular because it tests the long short-term memory of the system. An example dataset from this task is presented in Figure 10. The length of the sequence is given by T. a_1, a_2, a_3 and a_4 are the input signals, and y_1, y_2 and y_3 are the output signals. At each time step t only one input signal and one output signal can have the value 1. The values of a_1 and a_2 at the first five time steps give the pattern that the system will learn. The next T_d time steps represent the distractor period, where the system is distracted from the previous inputs. This is done by setting the value of a_3 to 1. After the distractor period, the a_4 signal is fired, which marks the cue signal. The system is then asked to repeat the input pattern on the outputs y_1 and y_2. The output y_3 is awaiting a signal, which is supposed to be 1 right until the input pattern is repeated. More details on the five-bit memory task can be found in [30].

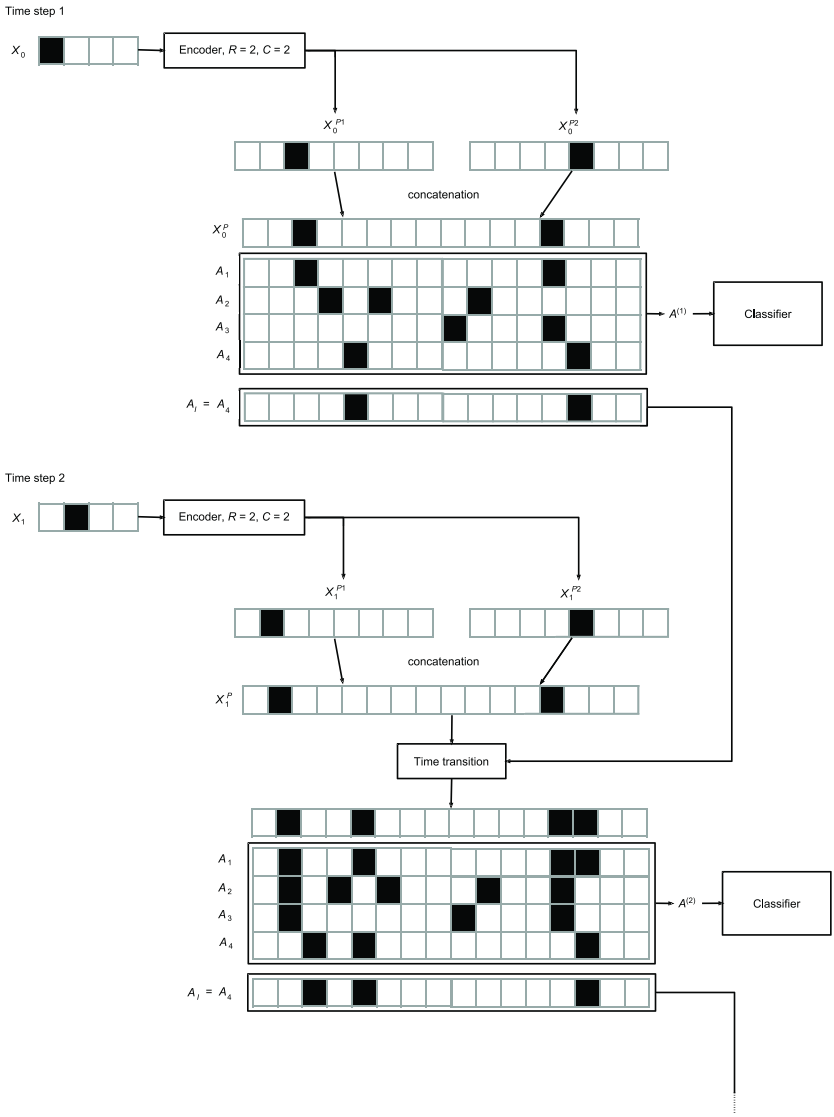

Figure 8. Architecture of the implemented system. The encoding is done according to the encoding scheme as shown in Figure 5, but with the slight modification of the C parameter. The encoding is exemplified with $R = 2$ and $C = 2$, which yields a size of eight for each permutation. The two permutations are then concatenated. At time step 1, there are no previous inputs, and the concatenated vector is simply used as the first iteration of the CA reservoir. The rule Z is then applied for I iterations. At time step 2, the encoding and concatenation are repeated. The time transition scheme is then applied, as described in Figure 7. The procedure as described in time step 2 is repeated until the end of the sequence.

https://doi.org/10.25088/ComplexSystems.26.3.225

Figure 9. Example run of the ReCA system with rule 90. The run is done with the parameters $R = 8$, $I = 4$ and $C = 5$. The horizontal gray lines represent a time step, in which the time transition function is applied to every bit. Time flows downward. The visualization is produced with the ReCA system described in this paper.

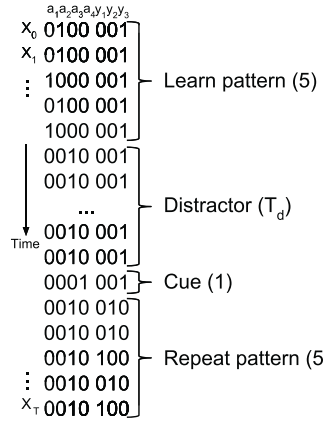

Figure 10. Example data from the five-bit task. The length of the sequence is T. The signals a_1, a_2, a_3 and a_4 are input signals, while y_1, y_2 and y_3 are output signals. In the first five time steps the system learns the pattern. The system is then distracted for T_d time steps. After the cue signal is set, the system is expected to reproduce the pattern that was learned.

3.1 Use of Parallel Cellular Automaton Reservoirs

In this paper the use of parallel reservoirs is proposed. The concept is shown in Figure 11. At the boundary conditions, that is, the cell at the very end of the reservoir, the rule will treat the cell that lies within the other reservoir as a cell in its own reservoir. This causes information/computation to flow between the reservoirs (loosely coupled).

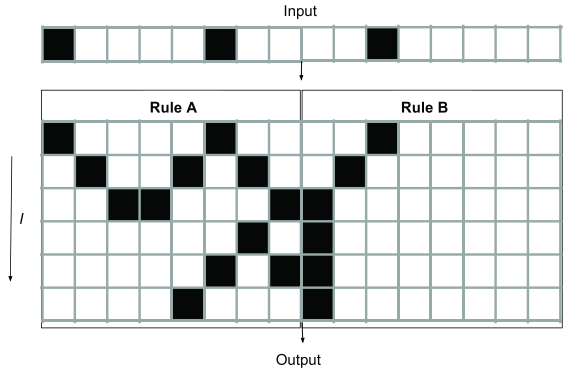

Figure 11. Concept behind parallel CA reservoirs. Iterations flow downward. The rules are interacting at the middle boundaries and at the side boundaries, where the CA wraps around.

By having different rules in the reservoirs, it might be possible to solve different aspects of the same problem, or even two problems at the same time. In [8], both the temporal parity and the temporal density task [30] are investigated.

Which rule is most suited for a task is still an open research question. The characteristics and classes described in Section 2.3 are useful knowledge; however, they do not precisely describe why some rules perform better than others on different tasks. In Figure 12, an example run of the parallel system is shown, with rule 90 on the left and rule 182 on the right. This visualization gives useful insights on how the rules interact.

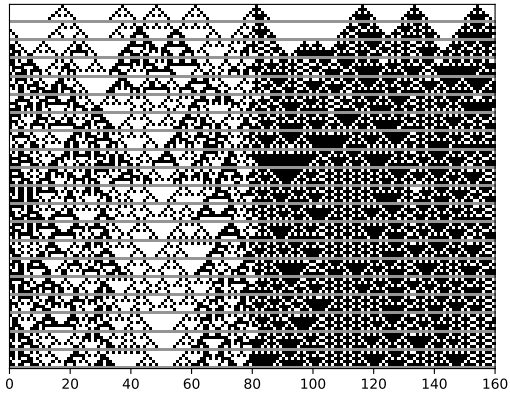

Figure 12. Example run of the ReCA system with rule 90 on the left and rule 182 on the right. Information is allowed to flow between the reservoirs. The run is done with the parameters $R = 8$, $I = 4$ and $C = 5$. The horizontal gray lines represent a time step, in which the time transition function is applied to every bit. Time flows downward. The visualization is produced with the implemented system.

https://doi.org/10.25088/ComplexSystems.26.3.225

3.2 Measuring Computational Complexity of a Cellular Automaton Reservoir

The size of the reservoir is crucial for the success of the system. In this paper, the reservoir size is measured by $R * I * C$. As seen in Section 3.1, the size of the reservoirs will remain the same both for the one-rule reservoirs and the two-rule reservoirs. This is crucial in order to be able to directly compare their performances.

4. Results

The parameters for the five-bit memory task used can be seen in Table 1. The same parameters as in the single-reservoir system are used in the quasi-uniform CA reservoir system with a combination of two rules. The tested combinations of rules are shown in Table 2.

Training set size	32
Testing set size	32
Distractor period	200
Number of runs	120

Table 1. Five-bit task parameters.

CA rules	60, 90, 102, 105, 150, 153, 165, 180, 195
I (iterations)	2, 4
R (random mappings)	4, 8
C (size multiple)	10

Table 2. CA reservoir parameter combinations.

4.1 Results from the Single-Reservoir Computing System

The results from the single-reservoir ReCA system can be seen in Table 3. The results in this paper are significantly better than what was reported in [8]. We can, however, see a similar trend. Rules 102 and 105 were able to give promising results, while rule 180 was not very well suited for this task. Exceptions are rules 90 and 165, where the results in Table 3 show very high accuracy. In [7] very promising results from rule 90 are also achieved.

4.2 Results from the Parallel (Nonuniform) Reservoir Computing System

Results can be seen in Table 4. It can be observed that rules that were performing well in Table 3 seem to give good results when combined. However, some combinations of rules, for example, 60 and 102, 153 and 195, gave worse results than the rules by themselves. We can observe the same tendencies as in the single runs: higher R and I generally yield better results.

Rule	$I = 2, R = 4$	$I = 2, R = 8$	$I = 4, R = 4$	$I = 4, R = 8$
60	25.8%	53.3%	76.7%	95.0%
90	100.0%	100.0%	97.5%	100.0%
102	30.8%	63.3%	71.7%	96.7%
105	95.8%	99.2%	99.2%	100.0%
150	96.7%	100.0%	100.0%	100.0%
153	26.7%	55.0%	80.0%	95.0%
165	100.0%	100.0%	100.0%	100.0%
180	9.2%	38.3%	0.8%	1.7%
195	39.2%	61.7%	79.2%	95.8%

Table 3. Single-reservoir CA on the five-bit task. Successful runs with $T = 200$.

Rule	$I = 2, R = 4$	$I = 2, R = 8$	$I = 4, R = 4$	$I = 4, R = 8$
60 and 90	87.5%	100.0%	96.9%	100.0%
60 and 102	0.0%	0.0%	0.0%	0.0%
60 and 105	81.2%	100.0%	96.9%	100.0%
60 and 150	71.9%	100.0%	96.9%	100.0%
60 and 153	0.0%	0.0%	0.0%	0.0%
60 and 165	87.5%	93.8%	96.9%	96.9%
60 and 180	43.8%	53.1%	90.6%	84.4%
60 and 195	0.0%	0.0%	0.0%	0.0%
90 and 102	90.6%	100.0%	100.0%	96.9%
90 and 105	100.0%	100.0%	100.0%	100.0%
90 and 150	100.0%	100.0%	100.0%	100.0%
90 and 153	93.8%	96.9%	96.9%	100.0%
90 and 165	90.6%	100.0%	100.0%	100.0%
90 and 180	90.6%	100.0%	100.0%	100.0%
90 and 195	87.5%	96.9%	100.0%	100.0%
102 and 105	78.1%	100.0%	96.9%	100.0%
102 and 150	81.2%	100.0%	96.9%	100.0%
102 and 153	0.0%	0.0%	0.0%	3.1%
102 and 165	93.8%	100.0%	100.0%	100.0%
102 and 180	0.0%	40.6%	3.1%	6.2%
102 and 195	0.0%	0.0%	0.0%	3.1%
105 and 150	93.8%	100.0%	100.0%	100.0%
105 and 153	75.0%	93.8%	93.8%	100.0%
105 and 165	96.9%	100.0%	100.0%	100.0%
105 and 180	93.8%	100.0%	100.0%	100.0%
105 and 195	65.6%	93.8%	96.9%	100.0%
150 and 153	87.5%	100.0%	96.9%	100.0%
150 and 165	100.0%	100.0%	100.0%	100.0%
150 and 180	81.2%	100.0%	100.0%	100.0%
150 and 195	78.1%	96.9%	100.0%	100.0%
153 and 165	81.2%	100.0%	100.0%	100.0%

Table 4. (*continues*)

https://doi.org/10.25088/ComplexSystems.26.3.225

Rule	$I = 2, R = 4$	$I = 2, R = 8$	$I = 4, R = 4$	$I = 4, R = 8$
153 and 180	3.1%	46.9%	0.0%	0.0%
153 and 195	0.0%	0.0%	0.0%	0.0%
165 and 180	96.9%	96.9%	100.0%	100.0%
165 and 195	87.5%	100.0%	100.0%	100.0%
180 and 195	40.6%	87.5%	93.8%	96.9%

Table 4. Parallel CAs on the five-bit task. Successful runs with $T = 200$.

5. Analysis

5.1 Single-Reservoir Computing System

The complexity of the reservoir is a useful metric when comparing different approaches. If we examine rule 90, we can observe that it achieves a 100% success rate at $I = 4$, $R = 8$ and $C = 10$. The size of the reservoir is $4 * 8 * 10 = 320$ at this configuration. Note that even though lower values of R and I also give 100%, at $R = 4$ and $I = 4$ the success is 97.5%, and again 100% at $I = 4$ and $R = 8$. A 100% success rate on the same task was reported in [7] with $R = 32$ and $I = 16$. The C parameter was set to 1. As such, the size of the reservoir is $32 * 16 * 1 = 512$ (feedforward architecture).

Results on the five-bit task using the recurrent architecture were also presented in [7]. A 100% success rate was achieved with $I = 32$ and $R = 45$. This yields a reservoir size of $32 * 45 = 1440$. Those results were intended to study the relationship between the distractor period of the five-bit task and the R number of random mappings. The I was kept fixed at 32 during this experiment. Even if the motivations for the experiments were different, the comparison of results gives insight that the reservoir size itself may not be the only factor that determines the performance of the ReCA system.

5.2 Parallel (Nonuniform) Reservoir Computing System

Why are some combinations better than others? As observed in Section 4, rules that are paired with other rules that perform well on their own also perform well together. The combination of rule 90 and rule 165 is observed to be very successful. As described in [38], rule 165 is the complement of rule 90. If we observe the single-CA results in Table 3, we can see that rules 90 and 165 perform very similarly.

Examining one of the worst-performing rule combinations of the experiments, that is, rule 153 and rule 195, we get a useful insight, as seen in Figure 13. Here it is possible to notice that the interaction of rules creates a "black" region in the middle (between the rules), thereby effectively reducing the size of the reservoir. As described in [39], rules 153 and 192 are mirrored complements.

Rule 105 is an interesting rule to be combined with others. As described in [29], the rule does not have any complements or any mirrored complements. Nevertheless, as seen in Table 2, it performs well in combination with most other rules.

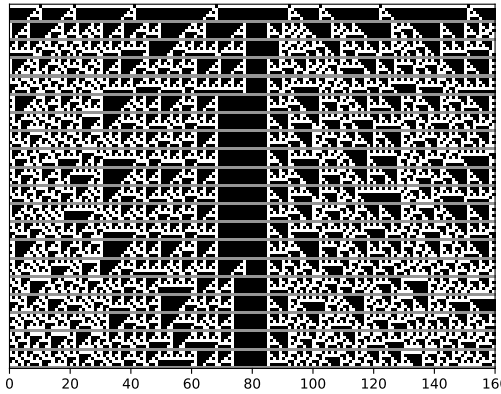

Figure 13. Example run of the ReCA system with rules 153 and 195. The run is done with the parameters $R = 8$, $I = 4$ and $C = 5$. The horizontal gray lines represent a time step, in which the time transition function is applied to every bit. Time flows downward. The visualization is produced with the implemented system.

6. Conclusion

A framework for using cellular automata (CAs) in reservoir computing has been implemented, which makes use of uniform CAs and quasi-uniform CAs. The relationship between reservoir size and performance of the system are presented. The implemented configuration using a parallel nonuniform cellular automaton (CA) reservoir is tested in this paper for the first time (to the best of the authors' knowledge). Results have shown that some CA rules work better in combination than others. Good combinations tend to have some relation—for example, being complementary. Rules that are mirrored complements do not work well together, because they effectively reduce the size of the reservoir. The concept is still very novel, and a lot of research is left to be done, regarding both the use of a nonuniform CA reservoir and reservoir computing using cellular automata (ReCA) systems in general.

As previously discussed, finding the best combination of rules is not trivial. If we only consider the usage of two distinct rules, the rule space grows from only 256 single-reservoir options to

$256!/2! * 254! = 32\,640$ different combinations. Matching two rules that perform well together can be quite a challenge. By investigating the characteristics of the rules, for example, with a lambda parameter [14], Lyapunov exponent [40] or other metrics, it may be possible to pinpoint promising rules. Ideally, the usage of more than two different rules could prove a powerful tool. The rule space would then grow even larger, and an exhaustive search would be infeasible. However, one possibility (that we are currently investigating) is to use evolutionary algorithms to search for suitable rules. Adding more and more rules would bring the reservoir closer to a true nonuniform CA.

In [30] a wide range of different tasks is presented. In this paper, only one (five-bit task) is used as a benchmark. By combining different rules' computational power, a nonuniform CA reservoir could be designed that performs well on a variety of tasks.

References

[1] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, Cambridge, MA: The MIT Press, 2016.

[2] Y. Bengio, P. Simard and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent Is Difficult," *IEEE Transactions on Neural Networks*, **5**(2), 1994 pp. 157–166. doi:10.1109/72.279181.

[3] H. Jaeger, *The "Echo State" Approach to Analysing and Training Recurrent Neural Networks—with an Erratum Note*, GMD Technical Report, 148:34, Bonn, Germany: German National Research Center for Information Technology, 2001. (Jul 12, 2017) minds.jacobs-university.de/sites/default/files/uploads/papers/EchoStatesTechRep.pdf.

[4] T. Natschläger, W. Maass and H. Markram, "The 'Liquid Computer': A Novel Strategy for Real-Time Computing on Time Series," *Special Issue on Foundations of Information Processing of Telematik*, **8**(1), 2002 pp. 39–43.

[5] J. Von Neumann, *Theory of Self-Reproducing Automata* (A. W. Burks, ed.), Urbana, IL: University of Illinois Press, 1966 pp. 3–14.

[6] O. Yilmaz, "Reservoir Computing Using Cellular Automata." arxiv.org/abs/1410.0162.

[7] O. Yilmaz, "Connectionist-Symbolic Machine Intelligence Using Cellular Automata Based Reservoir-Hyperdimensional Computing." arxiv.org/abs/1503.00851.

[8] E. T. Bye, "Investigation of Elementary Cellular Automata for Reservoir Computing," Master's thesis, NTNU, Norway, 2016. brage.bibsys.no/xmlui/handle/11250/2415318.

[9] M. Margem and O. Yilmaz, "An Experimental Study on Cellular Automata Reservoir in Pathological Sequence Learning Tasks." (Jul 12, 2017) ozguryilmazresearch.net/Publications/MargemYilmaz_TechReport2016.pdf.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)*, Boston, MA, IEEE, 2015 pp. 1–9. ieeexplore.ieee.org/document/7298594.

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, **529**(7587), 2016 pp. 484–489. doi:10.1038/nature16961.

[12] M. Lukoševičius, H. Jaeger and B. Schrauwen, "Reservoir Computing Trends," *KI-Künstliche Intelligenz*, **26**(4), 2012 pp. 365–371. doi:10.1007/s13218-012-0204-5.

[13] N. Bertschinger and T. Natschläger, "Real-Time Computation at the Edge of Chaos in Recurrent Neural Networks," *Neural Computation*, **16**(7), 2004 pp. 1413–1436. doi:10.1162/089976604323057443.

[14] C. G. Langton, "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation," *Physica D: Nonlinear Phenomena*, **42**(1–3), 1990 pp. 12–37. doi:10.1016/0167-2789(90)90064-V.

[15] T. E. Gibbons, "Unifying Quality Metrics for Reservoir Networks," in *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN)*, Barcelona, Spain, IEEE, 2010 pp. 1–7. doi:10.1109/IJCNN.2010.5596307.

[16] C. Fernando and S. Sojakka, "Pattern Recognition in a Bucket," in *Proceedings of Advances in Artificial Life (ECAL 2003)*, Dortmund, Germany (W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich and J. T. Kim, eds.), Berlin, Heidelberg: Springer, 2003 pp. 588–597. doi:10.1007/978-3-540-39432-7_63.

[17] B. Jones, D. Stekel, J. Rowe and C. Fernando, "Is There a Liquid State Machine in the Bacterium *Escherichia coli*?" in *Proceedings of the IEEE Symposium on Artificial Life 2007 (ALIFE'07)*, Honolulu, HI, IEEE, 2007 pp. 187–191. doi:10.1109/ALIFE.2007.367795.

[18] D. Nikolić, S. Haeusler, W. Singer and W. Maass, "Temporal Dynamics of Information Content Carried by Neurons in the Primary Visual Cortex," in *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'06)*, Cambridge, MA: MIT Press, 2006 pp. 1041–1048. dl.acm.org/citation.cfm?id=2976456.2976587.

[19] M. Dale, J. F. Miller and S. Stepney, "Reservoir Computing as a Model for *in-Materio* Computing," *Advances in Unconventional Computing: Volume 1: Theory* (A. Adamatzky, ed.), Cham, Switzerland: Springer International Publishing, 2017 pp. 533–571. doi:10.1007/978-3-319-33924-5_22.

[20] M. Dale, J. F. Miller, S. Stepney and M. A. Trefzer, "Evolving Carbon Nanotube Reservoir Computers," in *Unconventional Computation and Natural Computation (UCNC 2016)*, Manchester, UK, Cham, Switzerland: Springer International Publishing, 2016 pp. 49–61. doi:10.1007/978-3-319-41312-9_5.

[21] M. Dale, S. Stepney, J. F. Miller and M. Trefzer, "Reservoir Computing *in Materio*: An Evaluation of Configuration through Evolution," in *Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Athens, Greece, IEEE, 2016. doi:10.1109/SSCI.2016.7850170.

[22] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman and S. Massar, "Optoelectronic Reservoir Computing." arxiv.org/abs/1111.7219.

[23] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutiérrez, L. Pesquera, C. R. Mirasso and I. Fischer, "Photonic Information Processing beyond Turing: An Optoelectronic Implementation of Reservoir Computing," *Optics Express*, **20**(3), 2012 pp. 3241–3249. doi:10.1364/OE.20.003241.

[24] D. Snyder, A. Goudarzi and C. Teuscher, "Computational Capabilities of Random Automata Networks for Reservoir Computing," *Physical Review E*, **87**(4), 2013 042808. doi:10.1103/PhysRevE.87.042808.

[25] A. V. Burkow, "Exploring Physical Reservoir Computing Using Random Boolean Networks," Master's thesis, NTNU, Norway, 2016. brage.bibsys.no/xmlui/handle/11250/2417596.

[26] C. Gershenson, "Introduction to Random Boolean Networks." arxiv.org/abs/nlin/0408006.

[27] M. Sipper, "The Emergence of Cellular Computing," *Computer*, **32**(7), 1999 pp. 18–26. doi:10.1109/2.774914.

[28] M. Cook, "Universality in Elementary Cellular Automata," *Complex Systems*, **15**(1), 2004 pp. 1–40. www.complex-systems.com/pdf/15-1-1.pdf.

[29] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[30] H. Jaeger, *Long Short-Term Memory in Echo State Networks: Details of a Simulation Study*, Technical report No. 27, Jacobs University Bremen, 2012. minds.jacobs-university.de/sites/default/files/uploads/papers/2478_Jaeger12.pdf.

[31] M. Gardner, "The Fantastic Combinations of John Conway's New Solitaire Game 'Life'," *Scientific American*, **223**(4), 1970 pp. 120–123.

[32] D. Kleyko, S. Khan, E. Osipov and S.-P. Yong, "Modality Classification of Medical Images with Distributed Representations Based on Cellular Automata Reservoir Computing," in *Proceedings of the 2017 14th International Symposium on Biomedical Imaging (ISBI 2017)*, Melbourne, Australia, 2017, IEEE, 2017. doi:10.1109/ISBI.2017.7950697.

[33] N. McDonald, "Reservoir Computing and Extreme Learning Machines Using Pairs of Cellular Automata Rules." arxiv.org/abs/1703.05807.

[34] S. Nichele and A. Molund, "Deep Reservoir Computing Using Cellular Automata." arxiv.org/abs/1703.02806.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, **12**, 2011 pp. 2825–2830. www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

[36] M. S. Gundersen. "Specialization Project H2016." (Jul 13, 2017) github.com/magnusgundersen/spec.

[37] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, **9**(8), 1997 pp. 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[38] E. W. Weisstein. "Rule 90" from Wolfram MathWorld—A Wolfram Web Resource. mathworld.wolfram.com/Rule90.html.

[39] E. W. Weisstein. "Rule 60" from Wolfram MathWorld—A Wolfram Web Resource. mathworld.wolfram.com/Rule60.html.

[40] R. Legenstein and W. Maass, "Edge of Chaos and Prediction of Computational Performance for Neural Circuit Models," *Neural Networks*, **20**(3), 2007 pp. 323–334. doi:10.1016/j.neunet.2007.04.017.