Reservoir Computing with Complex Cellular Automata

Neil Babson
Christof Teuscher
Department of Electrical and Computer Engineering
Portland State University
PO Box 751
Portland, OR 97207-0751, USA

Reservoir computing (RC) is a computational framework in which a dynamical system, known as the *reservoir*, casts a temporal input signal to a high-dimensional space, and a trainable *readout layer* creates the output signal by extracting salient features from the reservoir. Several researchers have experimented with using the dynamical behavior of elementary cellular automaton (CA) rules as reservoirs. CA reservoirs have the potential to reduce the size, weight and power (SWaP) required to perform complex computation by orders of magnitude compared with traditional RC implementations. The research described in this paper expands this approach to CA rules with larger neighborhoods and/or more states, which are termed complex, as opposed to the elementary rules. Results show that some of these non-elementary cellular automaton rules outperform the best elementary rules at the standard benchmark five-bit memory task, requiring half the reservoir size to produce comparable results. This research is relevant to the design of simple, small, and low-power systems capable of performing complex computation.

Keywords: reservoir computing (RC); cellular automata (CAs); cellular automata based reservoirs (ReCAs)

1. Introduction

The foundations of reservoir computing (RC) are the independently proposed echo-state networks (ESNs) and liquid-state machines (LSMs), both of which use a randomly connected artificial recurrent neural network as the reservoir. Since the inception of the field, researchers have looked for ways to optimize the selection of reservoir construction parameters. Hierarchical reservoirs reimpose a degree of topological structure on reservoir connectivity by breaking the monolithic reservoir into loosely connected subreservoirs. The realization that dynamical systems other than neural networks could act as reservoirs has caused increasing interest in alternative reservoir substrates using biological, chemical and physical dynamical systems.

https://doi.org/10.25088/ComplexSystems.28.4.433

The field of cellular automaton–based reservoir computing (ReCA) uses cellular automaton (CA) rules as the dynamical reservoir. ReCA has focused so far on the 256 elementary one-dimensional CA rules that have two states and a neighborhood of size three. This paper expands ReCA to include one-dimensional CA rules with larger neighborhoods and more states. These CA rules, which are not part of the set of one-dimensional elementary rules, will be referred to as *complex*.

CA rule performance was tested on the five-bit memory task that is standard in ReCA research. More expressive rules were found that outperform any of the elementary rules, requiring a smaller CA reservoir. This reduces the amount of computation required to train the output layer and to operate the reservoir.

Our results were found to be task dependent. Complex CA rules were not found that could solve the temporal density and temporal parity benchmark tasks with greater accuracy than the most successful elementary rules.

1.1 Reservoir Computing

RC is a relatively new approach to machine learning in which the inner dynamics of a recurrently connected system, the *reservoir*, are harnessed to cast temporal inputs into a high-dimensional space, enhancing their separability. A *readout layer* generates the output from a linear combination of the states of reservoir nodes. Figure 1 shows the components of a reservoir computing system. The idea of reservoirs as a new type of architecture for recurrent neural networks (RNNs) was proposed independently in 2001, under the name echo-state networks (ESNs) [1], and in 2002 as liquid-state machines (LSMs) [2]. The recurrent connections of an RNN cycle information back to the internal nodes, allowing them to possess *state*, or memory, which makes them suitable for sequential tasks such as speech recognition. Unlike traditional neural networks, the internal weights between the nodes of the reservoir used in RC are not trained. Only the weights to the output, or readout, layer are trained, providing a substantial reduction in the amount of computation required for learning.

A reservoir capable of representing the inputs in its internal dynamics can perform multiple computation tasks, even simultaneous tasks, by training different readout layers to extract the output. In both the original ESN and LSM reservoir design, nodes are connected at random, but as reservoirs found a growing number of successful applications, researchers examined alternate construction techniques [3] and showed that many types of systems besides RNNs produce effective reservoirs [4].

In order for a reservoir system to perform useful computation, it must possess the *echo-state property*, characterized by the term *fading memory*. The system has the ability to remember (or echo) inputs, but also forgets them over time. The *echo-state property* guarantees that the input driving the ESN will "wash out" the influence of the random initial condition of the reservoir, causing it to react predictably to inputs [1]. Dynamical systems operating at the "edge of chaos" between ordered and disordered behavior are believed to possess the highest computational power [5, 6].

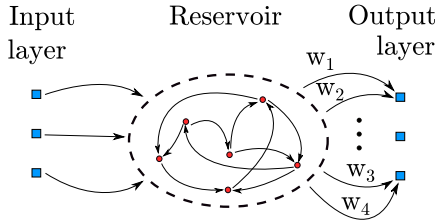

Figure 1. Components of a reservoir computing system. The input layer is connected to a subset of the reservoir nodes. The output layer is usually fully connected to the reservoir. Only the output weights w_i for $i \in \{1, \ldots, n\}$ are trained.

1.2 Cellular Automata

Cellular automata (CAs) are dynamical systems composed of discrete cells arranged in a grid of arbitrary dimension (usually one-, two- or three-dimensional), where each cell is in one of a finite number of states. At each *generation*, the cells are synchronously updated to a new state according to the CA transition rule, which is a function of the cell's previous state and that of its neighboring cells.

The CAs used in this paper are one dimensional, which means that a cell's neighborhood is a row of an odd number of contiguous cells, centered on itself and including the immediate neighbors to the left and right. Successive time steps are iterated downward to form a two-dimensional representation of the CA's evolution through time. The rule space of a CA depends on the size of the neighborhood N and the number of states S. The cell states are numbered from 0 to $S-1$. The number of possible neighborhood states is S^N and each of these may be mapped by the transition rule to one of the S states, giving a total rule space of S^{S^N}. A CA rule is used as a lookup table to apply the transition from each possible neighborhood state. Figure 2 illustrates how a CA rule is applied.

Wolfram systematically investigated the 256 one-dimensional rules with $S = 2$ and $N = 3$, which he named *elementary cellular automata* [7]. An elementary rule's number is found by reading the rule as a

https://doi.org/10.25088/ComplexSystems.28.4.433

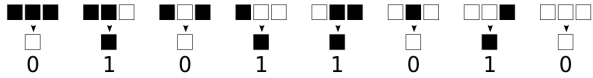

Figure 2. Elementary rule 90.

binary number and converting it to base 10. By convention, hexadecimal numbering is used for complex rules [8].

Wolfram also proposed a classification system based on the complexity of the emergent behavior of a CA rule. Class I CAs rapidly evolve to a homogeneous state from most initial configurations. Class II CAs evolve to a stable or simple periodic pattern. Class III rules lead to chaotic behavior without stable structures. In Class IV rules "edge of chaos" behavior can develop, where localized structures can last for long periods, interacting with each other in interesting and difficult-to-predict ways. An instance of a class IV rule, rule 110, has been proven to be Turing complete [9]. Figure 3 shows examples of the four classes.

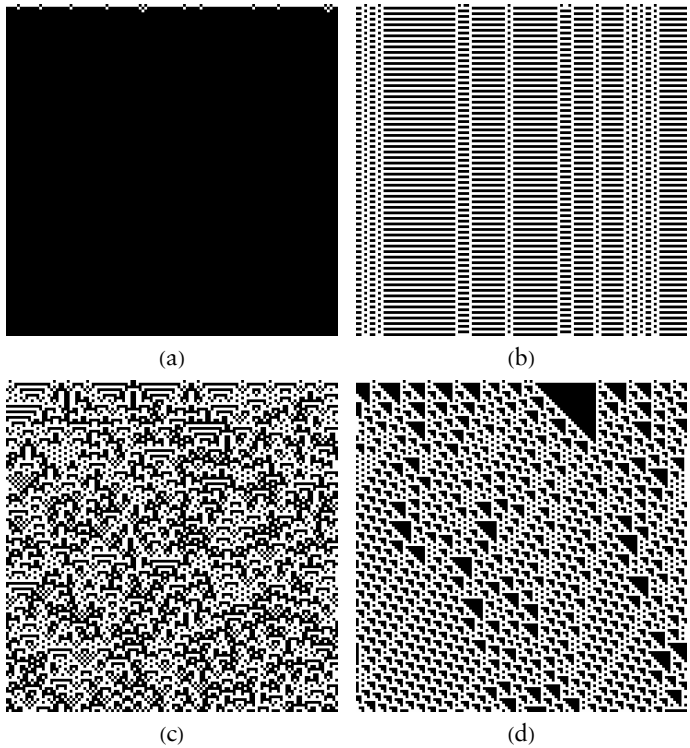

Figure 3. Wolfram's four classes of CAs represented by the elementary rules. (a) Class I: rule 251, (b) Class II: rule 1, (c) Class III: rule 105 and (d) Class IV: rule 193.

1.3 Previous Work

The use of elementary CA rules for RC was first proposed by Yilmaz in 2014 [10], showing that the framework was capable of solving memory tasks using orders of magnitude less computation than an ESN. The name ReCA was introduced in 2016 by Margem and Yilmaz [11]. Bye [12] investigated the performance of an ReCA system on the 30^{th}-order nonlinear autoregressive-moving-average (NARMA) benchmark, the temporal bit parity and temporal bit density tasks, as well as classification of vowel sound clips. Nonuniform elementary CA reservoirs were used to solve the five-bit memory task by Nichele and Gunderson in 2017 [13]. Also in 2017, Nichele and Molund [14] proposed a deep ReCA system using a two-layered reservoir. Kleyko et al. [15] demonstrated an ReCA system able to classify medical images with accuracy on par with traditional methods. McDonald [16] used pairs of elementary rules to implement an ReCA framework in which the reservoir requirements of hyperdimensional projection and short-term memory are explicitly separated into alternating modes of reservoir evolution. Morán et al. [17] demonstrated a hardware implementation of an ReCA system that performed pattern recognition on the handwritten numbers of the MNIST dataset.

2. Method

The ReCA system described in this section was implemented by the author in a C++ framework, which can be found at github.com/nbabson/CAreservoir. The architecture of the framework is similar to that used in [14] and [12].

2.1 Cellular Automata–Based Reservoirs System Design

The CA reservoir is made up of R subreservoirs, which receive identical temporal input signals. This technique of duplicating the reservoir has been used since the original ReCA paper and is found to be necessary for accurate results [10]. The leftmost cell of the first subreservoir is set to be the neighbor of the rightmost cell of the last subreservoir, creating a single circular CA. Within each subreservoir, a random mapping is generated between the elements of the input, of length L_n, and cells of the reservoir. The subreservoir size is known as the diffusion length L_d where $L_n < L_d$. The random mapping diffuses the inputs into the larger subreservoirs.

The reservoir is initialized with all the cells in the same state, either 0 for a two-state CA, or the highest numbered state if $S > 2$. The initial input overwrites select cell states, according to the mapping. For

https://doi.org/10.25088/ComplexSystems.28.4.433

applications with binary input, such as the five-bit memory benchmark and the temporal density and temporal parity benchmarks, this is done by replacing the initial state of the $R \cdot L_n$ input cells with 0 or 1. The reservoir processes the input by the application of the CA rule for I iterations, creating a CA reservoir of $R \cdot L_d \cdot (I+1)$ cells. The initial $R \cdot L_d \cdot I$ cells are vectorized to provide the input to the readout layer, while the last $R \cdot L_d$ cells form the initial CA state for the next time step, which is again selectively overwritten according to the input mapping. The rule is applied again, and the process repeats for each time step of the input data.

Figure 4 illustrates how the input is encoded into the reservoir and expanded by the CA rule to create the output vector. An alternative scheme for combining the inputs with the reservoir, similar to that adopted by Yilmaz in [18], adds the input value to the current cell state according to equation (1), where s^{t+1} is the state of the input cell s^t at the next time step after combining with the input bit i. Adding 1 to the resultant cell state ensures that every binary input affects the reservoir when $S > 2$. This approach was found to produce inferior results and is not used in this paper:

$$s^{t+1} = (s^t + i + 1) \bmod S. \tag{1}$$

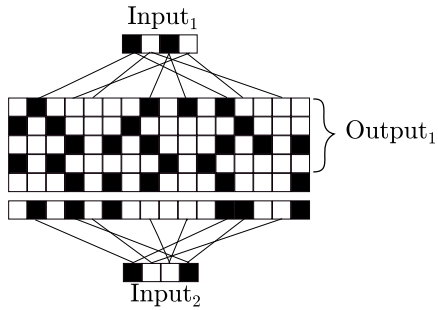

Figure 4. Mapping the input to the reservoir and generating the output. $L_n = 4$, $L_d = 8$, $R = 2$, $I = 4$, rule 90. Initially all the cells of the reservoir are in the same state. At each time step, the input is randomly mapped onto each of the R subreservoirs, overwriting the cell state. The CA rule is then applied I times. The last generation of the CA forms the initial state for the next time step, which is again overwritten according to the input mapping. The rest of the CA is vectorized and sent to the output nodes.

2.2 Readout Layer

The number of readout nodes the ReCA system requires is determined by the task being performed. Both of the benchmark tasks used in this

paper require the network to predict binary outputs at each time step. One readout node is needed for each output bit.

The weights from identical locations of the CA created at each time step are treated equally and used to predict output by reflecting the system's response to inputs. The output weights from the $R \cdot L_n \cdot I$ ReCA cells to the readout nodes are set using a linear regression model. The outputs from all time steps, as well as the target values, are sent to the linear regression model all at once for fitting.

After the weights are set, the task is run again and the system predicts the outputs. The real-valued output of the linear regression at the output nodes is binarized, with output smaller than 0.5 rounded to 0, and equal to or greater than 0.5 rounded to 1.

The ReCA system uses two different linear regression implementations, the linreg package from the C++ AlgLib library and the linear_model.LinearRegression class from the Python scikit-learn library. The two implementations produced equivalent results but the sci-kit functions ran faster, so sci-kit was used for the experiments in this paper. The system also allows the option of using support vector machines (SVM) from the C++ Torch machine learning library as the classifier. Nichele and Gunderson used SVM in their 2017 ReCA paper on nonuniform reservoirs [13].

3. Benchmark Tests

3.1 Five-Bit Memory Task

The five-bit memory task benchmark tests a network's long short-term memory capability and has been shown to be difficult for RNNs, including ESN [19, 20]. All of the literature on ReCA uses the five-bit task benchmark, so it is an appropriate first test of the capabilities of complex versus elementary CA reservoirs. The task has four temporal binary input signals, i_1, i_2, i_3 and i_4, and three binary outputs, o_1, o_2 and o_3. During the first five time steps of one run of the input sequence, the i_1 input is one of the 32 possible five-digit binary numbers, while i_2 is always $(i_1 + 1) \bmod 2$ (1 when $i_1 = 0$ and vice versa). This is the *message* that the system is supposed to remember. While the message is being input, $i_3 = i_4 = 0$.

This is followed by a *distractor period* of T_d time steps. Following convention in ReCA research, all tests in this paper were done with $T_d = 200$. On all time steps of the distractor period except the last, $i_1 = i_2 = 0$, $i_3 = 1$ and $i_4 = 0$. On the last step of the distractor period, $i_4 = 1$, giving the cue signal that it is time for the system to reproduce the pattern. Up until this point, the expected output is

https://doi.org/10.25088/ComplexSystems.28.4.433

$o_1 = o_2 = 0$ and $0_3 = 1$. For the remaining five time steps of the run, the output bits should be the same as the *message*, matching the first three input bits during the first five time steps. While the message is repeated, the inputs are the same as during the distractor period, $o_1 = o_2 = o_4 = 0$ and $o_3 = 1$. Table 1 illustrates one run of the five-bit task.

This series of $T_d + 10$ time steps is a single run of the test and is repeated for each of the 32 possible message inputs. To pass the five-bit task, the trained system must correctly predict the output bits for all steps of the task. With $T_d = 200$, that is $210 \cdot 32 \cdot 3 = 20\,160$ accurate predictions.

Time Step	Input				Output			
1	**1**	**0**	0	0	0	0	1	input message
2	**0**	**1**	0	0	0	0	1	
3	**0**	**1**	0	0	0	0	1	
4	**0**	**1**	0	0	0	0	1	
5	**1**	**0**	0	0	0	0	1	
6	0	0	1	0	0	0	1	distractor period
...	0	0	1	0	0	0	1	
204	0	0	1	0	0	0	1	
205	0	0	0	1	0	0	1	cue signal
206	0	0	1	0	**1**	**0**	0	repeat message
207	0	0	1	0	**0**	**1**	0	
208	0	0	1	0	**0**	**1**	0	
209	0	0	1	0	**0**	**1**	0	
210	0	0	1	0	**1**	**0**	0	

Table 1. Run 17 of 32 of the five-bit memory task.

3.2 Temporal Density and Temporal Parity

For both the temporal density and the temporal parity classification tasks, the reservoir receives a single input stream of bits. The reservoirs were tested on both tasks simultaneously, using the same input stream. These two tasks were used by Snyder et al. to test the performance of a random Boolean network (RBN) reservoir [21], and by Bye to test the performance of a nonuniform ReCA system [12].

The reservoir continuously evaluates the incoming bits over a window of the last n time steps, where n is an odd number. The temporal density output node is trained to return 1 if the input window contains more ones than zeros and return 0 otherwise. The temporal parity output node is trained to return whether the number of ones in the input window is odd or even. For the temporal density task, a delay of τ time steps was included between the input and the expected response.

The single input was mapped to L_{in} cells in each subreservoir. In agreement with previous work, optimal performance was found when the ratio $L_{in} / L_d = 1 / 2$ [12], which is used for the experiments in this paper. The system was trained and tested on randomly generated sets of length L_t time steps. In order to pass either task, the reservoir must make $L_t - n - \tau$ accurate predictions, as the reservoir output is ignored for the first $n + \tau$ time steps of the test set. Table 2 shows the system parameters used for all the temporal density and temporal parity results in this paper.

I	CA iterations	4
R	number of subreservoirs	12
L_d	diffusion length	40
L_{in}	input length	20
L_t	training set length	400
τ	delay length	2
n	window size	3

Table 2. System parameters used for the temporal density and temporal parity benchmark tasks.

4. Experiments

4.1 Five-Bit Memory Task

In order to establish a baseline for the performance of the complex CA rules on the five-bit memory task, the ReCA framework was tested on a selection of elementary CA rules that were found to be the most successful at the task in previous work [10, 12]. With $I = 4$ and $R = 8$, only four elementary rules were able to achieve zero error on the five-bit task. These four rules are equivalent, being either mirror images of each other and/or having their states reversed.

Increasing the reservoir size by adding subreservoirs or applying the rule more times improved accuracy, but came at a cost of increased processing time required to perform the linear regression. Table 3 shows the results for the most promising elementary rules at different combinations of I and R. Table 4 gives average CPU times to train and test elementary CA reservoirs of different sizes, measured using the C standard library `clock()` function. The CPU time can be seen to scale linearly with the size of the reservoir. The results obtained here were somewhat worse for the smaller reservoir of $(I, R) = (4, 8)$ than in previous work using a similar system design [10, 12, 13].

Finding more effective rules among complex CAs enables the task to be reliably completed with a smaller reservoir than is required by

an elementary CA, thereby reducing the amount of computation required. All of the experiments described in this section were performed with $(I, R) = (4, 8)$ and $L_d = 40$. Only rules that passed the five-bit task on their first run were saved for further testing. Since none of the elementary CA rules were able to pass the benchmark as much as 10% of the time, this means that many rules that are capable of outperforming any of the elementary rules were rejected.

Rule	$(I, R) = (4, 8)$	(8, 8)	(4, 12)
60	9	99	92
90	0	22	2
102	9	99	86
153	6	99	87
195	6	99	92
210	6	12	66

Table 3. Successful trials out of 100 runs of the five-bit task for elementary CA rules.

(I, R)	Time
(4, 8)	7.72 s
(4, 12)	11.53 s
(8, 8)	15.35 s

Table 4. CPU time in seconds to train and test a reservoir using elementary rule 60 on the five-bit task for different values of I and R. Times are averages of five test runs.

4.1.1 Three-State Cellular Automaton Reservoir

One-dimensional CAs with three states and a neighborhood of three have a transition function specified by a lookup table 27 digits long. The rule space is $3^{27} \approx 7.6 \cdot 10^{12}$. A stochastic search of the space randomly generated rules to test on the five-bit memory task. As the most computationally intensive part of the search is performing the linear regression on the reservoirs, class I and class II rules, whose behavior is believed to be too simple to support computation, are removed. The CAs are first evolved according to their rule without any inputs beyond a random initial configuration. Rules for CAs that settle into a uniform or oscillating state in the first 100 generations, as well as those that merely shift to the left or right, are removed from consideration.

The remaining rules are tested on the five-bit memory task. Those that succeed without any inaccurate predictions are saved for further testing. These are then scored on 100 runs of the five-bit task. Out of

1000 runs, rules that passed the benchmark were found 2.0% of the time, and 17.6% of the randomly generated rules were discarded as class I or II.

4.1.2 Neighborhood Five Cellular Automaton Reservoir

Two-state neighborhood five rules have a rule space of $2^{32} \approx 4.3 \cdot 10^9$. As with the three-state rules, this space was searched stochastically, with class I and class II rules ignored. Rules for reservoirs that pass the five-bit memory task were saved and tested on 100 further runs of the benchmark. Of 1000 runs, 2.2% of randomly generated rules passed the benchmark, and 16.2% of the rules were rejected before testing for being class I or class II.

4.1.3 Population Density Rules

Population density transition rules are a function of a cell's state and the count of how many cells are in each state $s \in S$ among its neighbors. For $(S, N) = (2, 5)$, these are a left-right symmetrical subset of the two-state neighborhood five rules. The transition function lookup table is 10 digits long, and each of the 1024 rules was investigated. Those exhibiting class I or class II behavior were rejected, while the rest were tested on the five-bit memory task. Nine rules passed the benchmark. Those that passed were saved and tested on the benchmark 100 times.

4.1.4 Evolving Complex Cellular Automaton Rules

Complex CAs with more than three states have a vast rule space. For $(S, N) = (3, 4)$ and $(S, N) = (3, 5)$ the number of possible CA rules is $\approx 3.4 \cdot 10^{38}$ and $\approx 2.4 \cdot 10^{87}$, respectively. These spaces are not amenable to a stochastic search for rules able to perform the five-bit memory task. Hundreds of randomly generated rules failed to produce any that were successful. Almost all of these reservoirs mapped every input to the most frequently occurring expected output for each output layer node. This is because the vast majority of these rules are class III, too chaotic to allow stable structures to persist and therefore unable to support computation.

In order to reduce the chaoticity, a genetic algorithm (GA) [22] was used to evolve rules more likely to have class III behavior. The very chaotic rules found by randomly generating a CA with $S > 3$ tend to have small contiguous regions in their two-dimensional spacetime representation, resembling television static. The GA fitness function selects for rules more likely to have class IV behavior by maximizing the size of these contiguous regions, which allows for the possibility of developing stable localized structures.

https://doi.org/10.25088/ComplexSystems.28.4.433

The GA algorithm starts with a randomly generated population of 32 rules. At each epoch, the rule is applied for 200 iterations to an initial random configuration of 400 cells. The fitness of each individual is evaluated using a "smallest-largest" rule. The largest contiguous region of cells belonging to the same state, adjacent horizontally or vertically, is identified for each state $s \in S$. The fitness assigned to the rule is equal to the size of the smallest of these largest connected areas. Using the area for the state whose contiguous region is smallest ensures that all the states contribute to the dynamic behavior of the CA rule and prevents the tendency for the GA to evolve toward a uniform class I behavior.

The least-fit half of the population is discarded and replaced by the next generation of offspring. The fit half of the population forms eight pairs that each produce two new rules by separating at a randomly chosen location and recombining, as seen in Figure 5. Each offspring receives a single random mutation changing one digit of its rule. Evolution continues for 200 generations or until the fitness of the best individual equals or exceeds 200, which rarely takes more than 150 generations.

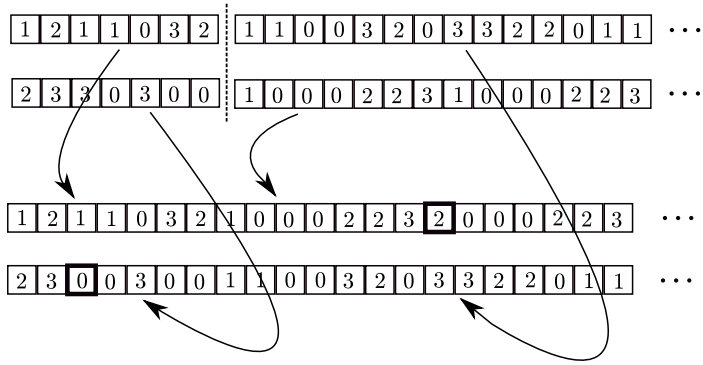

Figure 5. Two rules produce a pair of children by splitting at a randomly chosen point and recombining. Each child receives a single random mutation.

Both the mating and the mutation portion of the algorithm were found to be necessary; with either one removed, evolution happens much more slowly, if at all. The fitness goal of at least 200, for both four-state and five-state rules, was chosen empirically as the point where the evolved rules were making the fewest incorrect predictions when given the five-bit task.

The evolved four-state rules passed the benchmark 2.5% of the time and the five-state rules passed 1.1%. Many more missed fewer than 10 predictions, which was not observed to happen with unevolved rules.

4.2 Temporal Density and Temporal Parity

The elementary rules found to be most successful at the parity task by Bye were tested 100 times on the joint benchmark task to serve as a baseline of comparison for the complex rule's performance. As with the five-bit task, three-state and neighborhood five CA rules were found by a stochastic search of the rule space. Rules that achieved 100% correct prediction on either the temporal density or the temporal parity task were saved for further testing. This occurred on 2.3% of the runs for three-state rules and 1.8% for neighborhood five rules.

Unlike with the previous benchmark, class I and class II rules were not removed from consideration before testing. This is because it was empirically observed that rules that perform well on the temporal density/parity tasks tend to exhibit a very low degree of chaoticity. Because of the high percentage of CA cells that were mapped as input for this benchmark, a rule that is class I or II when evolved without interference can exhibit complex dynamics when used as a reservoir. Figure 6 demonstrates this using elementary rule 48, the most successful at the combined benchmark.

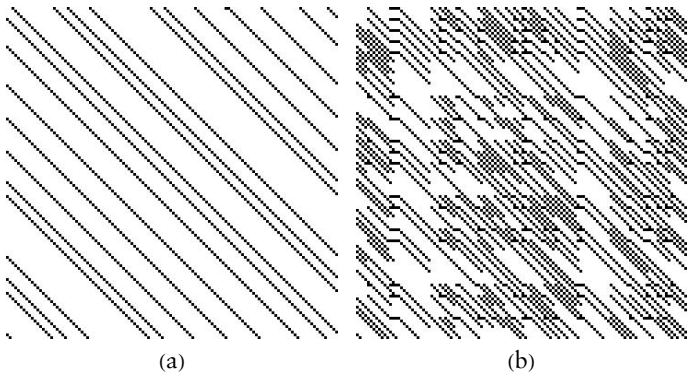

Figure 6. Elementary rule 48. (a) Evolved with only initial input, the rule produces a system lacking interacting structures. (b) Used as a reservoir for the temporal density and temporal parity benchmarks, the rule produces interacting structures and predicts the output without error.

Among the two-state neighborhood five population density rules, only one was found that got zero error on either of the tasks, so this class of rules was not investigated further for this benchmark.

To find four- and five-state CA rules, a GA was again used to reduce the chaoticity of contestant rules. Evolution continued until the smallest-largest fitness score reached 400. This is twice the fitness target for the five-bit task, again due to the lower chaoticity of rules that perform well on the temporal tasks.

https://doi.org/10.25088/ComplexSystems.28.4.433

4.3 Nonuniform Reservoir

A nonuniform CA is one in which the cells do not all implement the same rule. The ReCA framework used here implements a parallel nonuniform CA in which the reservoir is split in half into two subreservoirs using different rules. The cells at the boundaries between the two subreservoirs consider their neighbors in the other subreservoir to be the same as those in their own subreservoir in the application of the rule. This allows information to flow between the two regions. Previous work has shown that certain combinations of elementary rules in a parallel ReCA perform better than either of the rules individually at the five-bit memory task, while other combinations impede performance [13]. For the combined temporal density/parity task, Bye [12] found that only nonuniform automata could correctly predict 100% of the outputs.

In this paper, combinations of complex rules were tested together to see whether complementary rule combinations could be found for the different types of complex CA rules investigated.

5. Results

From each category of rules that were investigated, the six most successful rules at performing the five-bit benchmark task using reservoirs with parameters $(I, R, L_d) = (4, 8, 40)$ were selected for testing with smaller reservoirs. Due to the length of the complex rules, these were given labels. The best-performing rules and their labels are shown in Table 5.

Three-State Rules $(S, N) = (3, 3)$	
$S3_1$	212aa02ad02
$S3_2$	5ec2484e083
$S3_3$	34be5823dc1
$S3_4$	3d337739e3f
$S3_5$	3db9f53398b
$S3_6$	58db54c1a35
Neighborhood Five Rules $(S, N) = (2, 5)$	
$N5_1$	9b8dc760
$N5_2$	00cddf40
$N5_3$	725fb240
$N5_4$	fd3a6d12
$N5_5$	4f716154
$N5_6$	71ee3aaf

Table 5. (*continues*)

Density Rules $(S, N) = (2, 5)$	
D_1	13b
D_2	254
D_3	2d0
D_4	3ca
D_5	275
D_6	28d
Four-State Rules $(S, N) = (4, 3)$	
$S4_1$	b9aa502ddbabb5e8c5b715087a01da08
$S4_2$	e2604a35fca689cefc93a671b4689640
$S4_3$	1f31e800e11c86ba09bafbe00073d533
$S4_4$	bcbb4b3f7b915b47fd154586fa2d15a1
$S4_5$	53bf9dcb97e96e0247c9a980a1291d7e
$S4_6$	73838653ccc875124f5b4658516662c3
Five-State Rules $(S, N) = (5, 3)$	
$S5_1$	1871abdbbb1f751c5649ce2208abe0525e5e9
	8543e527293d8fe32c1cbec8f9493da900dc
$S5_2$	159b7bf409f219a835b7bbe69791bfa94432d
	737117268fff0b99044ac2ccbb513ca91456
$S5_3$	1b4cca207dfd49a79d60a0f0a36fda79a911d
	b434135e22cd79e6e04c69f086c7aab1712a
$S5_4$	1ca6bafcb8cc14e379373cb798461269c4d81
	258cc28720c74f12dd39b8e48d5863e176a9
$S5_5$	3ac662317b4445a8e63b425645228e27bfc4a
	0a388497b03a0adede89e87685b12977008f
$S5_6$	053b86edcbce84369a1eeb58079725743b506
	f228e9459e488139f0763489461f3847c82a

Table 5. The six best-performing rules on the five-bit memory task from each of the categories of complex CA rules investigated.

The selected rules were then further tested with a range of the reservoir parameters I, R and L_d. The results are shown in Table 6.

Rule	(4, 8, 40)	(4, 8, 30)	(4, 6, 40)	(4, 4, 40)	(3, 8, 40)	(4, 8, 20)
Elementary Rules $(S, N) = (2, 3)$						
60	9	0	0	0	0	0
90	0	0	0	0	0	0
102	9	0	0	0	0	0
153	6	0	0	0	0	0
195	6	0	0	0	0	0
210	6	0	0	0	0	0

Table 6. (*continues*)

Rule	(4, 8, 40)	(4, 8, 30)	(4, 6, 40)	(4, 4, 40)	(3, 8, 40)	(4, 8, 20)
Three-State Rules $(S, N) = (3, 3)$						
$S3_1$	100	100	100	0	25	10
$S3_2$	99	63	44	0	16	0
$S3_3$	100	92	89	2	0	6
$S3_4$	100	53	42	0	0	0
$S3_5$	100	85	93	2	2	0
$S3_6$	99	97	79	6	5	0
Neighborhood Five Rules $(S, N) = (2, 5)$						
$N5_1$	97	83	61	0	0	0
$N5_2$	99	92	60	0	0	2
$N5_3$	99	98	69	0	88	2
$N5_4$	95	75	59	0	0	0
$N5_5$	95	87	26	0	0	0
$N5_6$	93	49	46	35	38	0
Density Rules $(S, N) = (2, 5)$						
D_1	68	47	23	0	1	0
D_2	70	0	6	0	0	0
D_3	94	62	63	0	0	20
D_4	96	87	57	0	0	0
D_5	99	17	35	0	25	0
D_6	64	58	27	0	1	0
Four-State Rules $(S, N) = (4, 3)$						
$S4_1$	90	75	58	0	12	1
$S4_2$	90	84	56	0	9	0
$S4_3$	93	12	9	0	42	0
$S4_4$	95	35	21	0	0	0
$S4_5$	100	85	66	0	11	1
$S4_6$	87	20	2	0	0	0
Five-State Rules $(S, N) = (5, 3)$						
$S5_1$	95	50	42	8	15	0
$S5_2$	95	76	70	0	7	0
$S5_3$	99	74	51	0	24	0
$S5_4$	80	1	9	0	2	0
$S5_5$	84	17	7	0	2	0
$S5_6$	77	13	6	0	5	0

Table 6. Successful runs out of 100 on five-bit task for elementary CAs and five types of complex CA rules. Column headings give the values for the parameters (I, R, L).

For the temporal density and temporal parity tasks, all of the rules were tested using the system parameters seen in Table 2. The labels for the most successful rules in each category are in Table 7, and the results are found in Table 8.

Three-State Rules $(S, N) = (3, 3)$	
$S3_1$	43eaeb0ea68
$S3_2$	688e5c88a95
$S3_3$	41d9be81e58
$S3_4$	41f81666893
$S3_5$	32d7be7347b
$S3_6$	58db54c1a35
Neighborhood Five Rules $(S, N) = (2, 5)$	
$N5_1$	000c040f
$N5_2$	ff19e808
$N5_3$	1d5d1f7d
$N5_4$	0073ebbf
$N5_5$	fdfc1734
$N5_6$	5d17350f
Four-State Rules $(S, N) = (4, 3)$	
$S4_1$	be2614f5ae52e0bceb0564a44a818504
$S4_2$	dd8a5ec44a19cf885d83514504883000
$S4_3$	5c9e5700e4a3423d37456595f56282c0
$S4_4$	ccb14db2d37a6039165a6d45151f4010
$S4_5$	17a2163c27560aaece871545a3208890
$S4_6$	754f9cb2174a865262d53515310f1120
Five-State Rules $(S, N) = (5, 3)$	
$S5_1$	094edc5925c6beb872f42363fd4499effd7a0 6330439d82cd00064a42299880a06a6300f7
$S5_2$	2e9ffae0f04c5a7902a36adca32e1aad8a22e 5486eb074b92abcb17ef0e97f22ef3812276
$S5_3$	4a09ab57fa452402cda01dd88d4aa08f24b83 f51a5c6a638ba703cbfc0e6d485e7cb98b3f
$S5_4$	04572ec6156d9dab4f6db2cda2bb51a94877d d8d554d71deb9e25698ec08cc0a95ca9ef65
$S5_5$	2c2b5702c45a01eccf1b8f8e29f2c4199f2ea 0fa00c9c4e67e2579d1c7f20a770182cca6c
$S5_6$	3e5309a22ea94f23fd18365607c9c64e7201f a39ea33e4b7543332f59c6d43ed51317bbd5

Table 7. The six best-performing rules on the combined temporal density/parity task from each CA rule category investigated.

https://doi.org/10.25088/ComplexSystems.28.4.433

Rule	Combined	TD	TP
Elementary Rules $(S, N) = (2, 3)$			
27	50	98	51
38	57	72	72
39	45	94	48
48	67	83	76
83	51	94	56
174	57	98	58
Three-State Rules $(S, N) = (3, 3)$			
$S3_1$	50	58	82
$S3_2$	48	78	61
$S3_3$	46	61	73
$S3_4$	45	66	67
$S3_5$	43	81	53
$S3_6$	42	77	55
Neighborhood Five Rules $(S, N) = (2, 5)$			
$N5_1$	50	96	52
$N5_2$	52	77	69
$N5_3$	43	88	52
$N5_4$	42	72	57
$N5_5$	34	44	66
$N5_6$	30	40	75
Four-State Rules $(S, N) = (4, 3)$			
$S4_1$	34	65	55
$S4_2$	33	63	54
$S4_3$	29	56	56
$S4_4$	24	40	57
$S4_5$	24	39	60
$S4_6$	23	57	46
Five-State Rules $(S, N) = (5, 3)$			
$S5_1$	29	56	45
$S5_2$	19	34	55
$S5_3$	16	36	52
$S5_4$	16	38	38
$S5_5$	11	16	70
$S5_6$	9	15	48
Nonuniform Rule			
$S3_1 + N5_1$	54	78	69

Table 8. Successful runs out of 100 on the temporal density and temporal parity benchmarks, as well as performing both tasks in tandem.

6. Discussion

The qualities that cause a CA rule to produce a good reservoir were not the same for the two benchmarks investigated, and none of the rules that worked well for one benchmark showed particular success at the other. The increased information that can be held by a complex rule proved helpful for the five-bit memory task. As can be seen in Figure 7, the most successful reservoir for this task falls into one of a number of possible periodic attractors after each message is input. The particular basin of attraction encodes the message, which is output when the cue signal is given. In contrast, complex rules were not found to outperform the best elementary rules at the temporal density and temporal parity tasks.

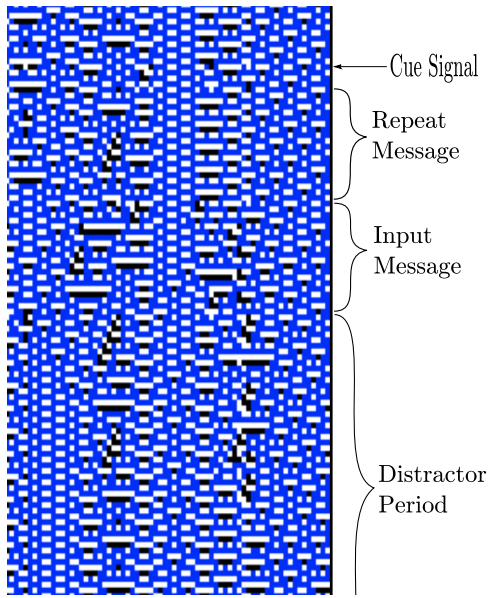

Figure 7. A segment of the history of a reservoir using rule $S3_1$ on the five-bit task with $I = 4$, $R = 6$ and $D_L = 30$. Each four horizontal lines is one reservoir time step. At the top of the image is the end of the distractor period after the input message 00000. The reservoir has fallen into a stable repeating pattern with a period of eight generations. After the cue signal is received, the reservoir "remembers" the message that was encoded in the repeating pattern. After the next message 00001 is input, the reservoir quickly falls into another stable pattern that encodes the new message.

6.1 Five-Bit Memory Task

While the best-performing elementary rules required a reservoir of size $I \cdot R \cdot L_d = 8 \cdot 8 \cdot 40 = 2560$ cells in order to achieve 99% success on the five-bit memory task, each of the five categories of complex

https://doi.org/10.25088/ComplexSystems.28.4.433

CA rules investigated here produced a rule capable of achieving 99% or 100% success on a reservoir of half that size, $I \cdot R \cdot L_d = 4 \cdot 8 \cdot 40 = 1280$ cells. The best-performing rules that were discovered were three-state neighborhood three. One of these was able to produce 100% correct results using a reservoir of size $I \cdot R \cdot L_d = 4 \cdot 6 \cdot 40 = 960$ cells. Only the neighborhood five population density rules were investigated exhaustively. For the other rule categories, continued searching would yield a virtually infinite number of similarly successful rules.

Rule performance correlates broadly with reservoir size, but the ability of the ReCA system to pass the benchmark falls at different rates, depending on which of the reservoir parameters I, R or L_d is lowered. The 30 complex rules average 91.7% correct test runs on the 1280-cell reservoir. Reducing the reservoir size by $1/4$ by lowering the diffusion length to 30 caused the least performance degradation, with the average success rate at 56.6%. Reducing the reservoir size by cutting the number of subreservoirs from eight to six had a slightly larger effect on the average reservoir performance, which fell to 45.9%. Finally, when the reservoir size was reduced to 960 cells by lowering the number of iterations I from four to three, the average success rate was only 11.0%, with 11 rules achieving zero correct runs, while a couple of rules were relatively resilient to the reduced iterations. When the reservoir size was halved from 1280 to 640 cells, the majority of the complex rules were unable to complete any successful runs of the task, and the average success rate was only 1.9%.

Figure 8 plots the average performance by reservoir category for different sizes of reservoir. At 1280 cells, the worst-performing category of complex rule passed the task on 88.2% of trials compared to 6.0% for the elementary rules. For reservoirs smaller than 1280 cells, elementary rules achieved 0.0% success.

An attempt was made to find pairs of rules that would complement each other in a nonuniform reservoir, performing better than either rule would alone, as has been shown to sometimes occur with pairs of elementary rules [13]. The six rules of each complex rule type can form 75 combinations of same-type rules, each of which can be used as a nonuniform reservoir for any setting of the reservoir size parameters. Of these possibilities, 20 combinations were tested with 100 runs of the benchmark. Pairs of rules were selected that performed well individually with the same reservoir parameters. None of these pairs performed better than the more successful of the two in a uniform reservoir, 11 of the pairings performing significantly worse than either rule tested alone. While it seems very likely that complex CA nonuniform pairings exist that do work well together, it remains an open question how common or rare they are.

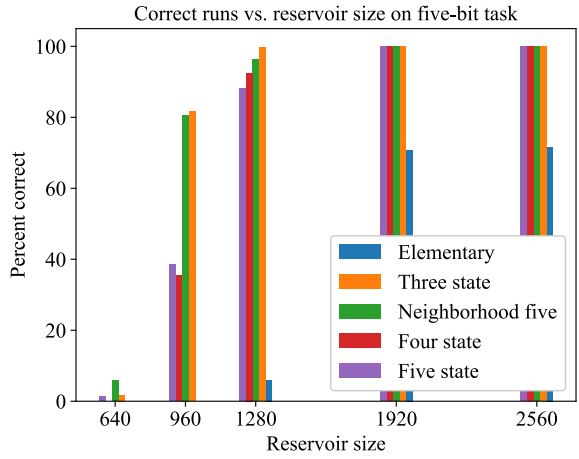

Figure 8. Five-bit memory task. Averaged performance of the six most successful rules in each category versus size of the reservoir in number of cells.

6.2 Temporal Density and Temporal Parity

For the combined temporal density and temporal parity task, the introduction of complex rules did not improve reservoir performance over the most successful elementary rules, demonstrating that the potential for time and power savings with complex rules depends on the requirements of the particular task. One nonuniform rule was found that combined two different categories of complex rules in a single reservoir that outperformed either of the constituent rules alone.

7. Conclusion

A reservoir computing with the cellular automata (ReCA) framework has been implemented, which expands the field of ReCA research from the 256 elementary cellular automaton (CA) rules to investigate rules with larger neighborhoods and more states, here called *complex* CA rules. A genetic algorithm (GA) was used to reduce the chaoticity of the four-state and five-state rule space, in order to find *edge of chaos* rules capable of computing the five-bit memory task benchmark.

The six best rules from each of five categories of complex rules were tested and shown to require half of the reservoir size as the most successful elementary rules to produce comparable results. This reduction in reservoir size equals a saving in power and time when operating the reservoirs. These results suggest that complex CA rules hold promise for the development of low-power simple systems capable of performing other complex computational tasks.

https://doi.org/10.25088/ComplexSystems.28.4.433

Acknowledgment

This work was supported in part by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] H. Jaeger, *The "Echo State" Approach to Analysing and Training Recurrent Neural Networks—with an Erratum Note*, German National Research Center for Information Technology GMD Technical Report 148, Bonn, Germany, 2001.
pdfs.semanticscholar.org/8430/c0b9afa478ae660398704b11dca1221ccf22.pdf.

[2] W. Maass, T. Natschläger and H. Markram, "Real-Time Computing without Stable States: A New Framework for Neural Computation Based on Perturbations," *Neural Computation*, **14**(11), 2002 pp. 2531–2560. doi:10.1162/089976602760407955.

[3] M. Lukoševicius and H. Jaeger, *Overview of Reservoir Recipes*, Technical Report 11, School of Engineering Science, Jacobs University, Bremen, Germany, 2007.
nbn-resolving.org/urn:nbn:de:gbv:579-opus-1006674.

[4] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano and A. Hirose, "Recent Advances in Physical Reservoir Computing: A Review," *Neural Networks*, **115**, 2019 pp. 100–123. doi:10.1016/j.neunet.2019.03.005.

[5] 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.

[6] 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.

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

[8] A. Wuensche, "Classifying Cellular Automata Automatically: Finding Gliders, Filtering, and Relating Space-Time Patterns, Attractor Basins, and the Z Parameter," *Complexity*, **4**(3), 1999 pp. 47–66.

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

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

[11] M. Margem and O. Yilmaz, *An Experimental Study on Cellular Automata for Reservoir Computing*, Technical Report, 2017.

[12] E. T. Bye, "Investigation of Elementary Cellular Automata for Reservoir Computing," Master's thesis, Department of Computer and Information Science, Norwegian University of Science and Technology, 2016.

[13] S. Nichele and M. S. Gunderson, "Reservoir Computing Using Nonuniform Binary Cellular Automata," *Complex Systems*, **26**(3), 2017 pp. 225–245. doi.org/10.25088/ComplexSystems.26.3.225.

[14] S. Nichele and A. Molund, "Deep Learning with Cellular Automaton-Based Reservoir Computing," *Complex Systems*, **26**(4), 2017 pp. 319–339. doi:10.25088/ComplexSystems.26.4.319.

[15] 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 *14th IEEE International Symposium on Biomedical Imaging (ISBI 2017)*, Melbourne, Australia, 2017, Piscataway, NJ: IEEE, 2017 pp. 1053–1056. doi:10.1109/ISBI.2017.7950697.

[16] N. McDonald, "Reservoir Computing & Extreme Learning Machines Using Pairs of Cellular Automata Rules," in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, 2017, Piscataway, NJ: IEEE, 2017 pp. 2429–2436. doi:10.1109/IJCNN.2017.7966151.

[17] A. Morán, C. F. Frasser and J. L. Rosselló, "Reservoir Computing Hardware with Cellular Automata." arxiv.org/abs/1806.04932.

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

[19] 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.

[20] H. Jaeger, *Long Short-Term Memory in Echo State Networks: Details of a Simulation Study*, Technical Report No. 27, School of Engineering Science, Jacobs University, Bremen Germany, 2012.

[21] 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.

[22] M. Mitchell, J. P. Crutchfield and R. Das, "Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work," in *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EcVA '96)*, Moscow, Russia: Russian Academy of Sciences, 1996.