Reservoir Computing with Complex Cellular Automata

Neil Babson, Christof Teuscher

Portland State University, P.O. Box 751, Portland, OR 97207-0751, USA

Abstract

Reservoir Computing 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. A suitable reservoir must possess the property of *fading memory* in order to process inputs. The foundations of Reservoir Computing are the independently proposed Echo State Networks and Liquid State Machines, 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 sub-reservoirs. The realization that dynamical systems besides neural networks could act as reservoirs has caused increasing interest in alternative reservoir substrates using biological, chemical, and physical dynamical systems. Several researchers have experimented with using the dynamical behavior of elementary cellular automaton rules as reservoirs. This research described in this paper expands this approach to cellular automaton 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 5-bit memory task, requiring half the reservoir size to produce comparable results.

Keywords: Reservoir computing, Cellular automata

1. Introduction

The small field of Cellular Automaton based Reservoir Computing (ReCA) has focused so far on the 256 elementary one-dimensional Cellular Automaton (CA) rules which have two states and a neighborhood of size three. This work expands ReCA to include one dimensional CA rules with larger neighborhoods and more states. These rules that are not part of the set of one-dimensional elementary rules will be referred to as *complex* CA rules. CA rule performance is tested on the 5-bit memory task that is standard in ReCA research. More expressive rules that outperform any of the elementary rules require a smaller CA reservoir, reducing the amount of computation required to train the output layer and to operate the reservoir.

The remainder of the paper is organized as follows. Section 2 presents background information on Reservoir Computing and Cellular Automata. Section 3 outlines the ReCA system architecture used in the experiments. The benchmark task used to evaluate the CA rules is described in Section 4. Section 5 lists the complex

Email address: nbabson@pdx.edu (Neil Babson)

CA experiments performed. Sections 6 and 7 provide results and discussion. Future work is discussed in Section 8, and Section 9 concludes the paper.

2. Background

2.1. Reservoir Computing

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 a 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 system 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].

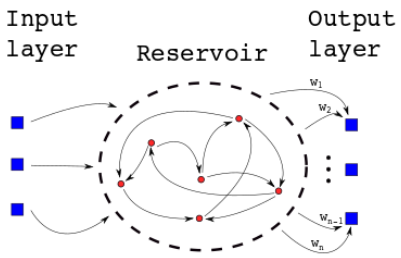

Figure 1: Components of a reservoir computing system. Input is connected to a subset of the reservoir nodes. Output is usually fully to the reservoir. Only the outout weights w_i for $i \in \{1, ..., n\}$ are trained.

2.2. Cellular Automata

Cellular Automata (CA) 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 CA 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 look-up table to apply the transition from each possible neighborhood state. Figure 2 illustrates how a CA rule is applied.

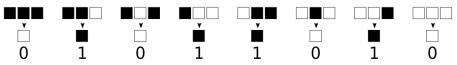

Figure 2: Elementary rule 90.

In his book *A New Kind of Science* 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 binary number and converting it to base-10. Similarly, when $S = 3$ the rule number is determined by converting the lookup table from base-3 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 an homogeneous state from most intial 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.

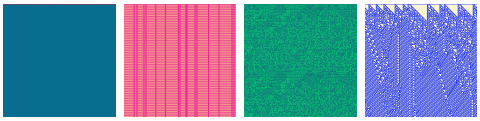

Figure 3: Wolfram's four classes of Cellular Automata rule represented by the elementary rules. (a) Class I: Rule 215, (b) Class II: Rule 1, (c) Class III: Rule 105, and (d) Class IV: Rule 193.

2.3. Previous Work

The use of elementary cellular automaton rules for reservoir computing was first proposed by Yilmaz in 2014, showing that the framework was capable of solving memory tasks using orders of magnitude less computation than an ESN [10]. The name ReCA was introduced in 2016 by Margem and Yilmaz [11]. Bye investigated the performance of a ReCA system on the 30th order nonlinear autoregressive-moving-average (NARMA) benchmark, the temporal bit parity and temporal bit density tasks, as well as classification of vowel sound clips [12]. Non-uniform elementary CA reservoirs were used to solve the 5-bit memory task by Nichele and Gunderson in 2017 [13]. Also in 2017 Nichele and Molund proposed a deep ReCA system using a two-layered reservoir. Kleyko et al. demonstrated a ReCA system able classify medical images with accuracy on par with traditional methods [14].

3. Method

The ReCA system described in this section was implemented by the author in a C++ framework which can be found at https://github.com/nbabson/CAreservoir. The architecture of the framework is similar to that uesd in [15] and [12].

3.1. ReCA System Design

The CA reservoir is made up of R sub-reservoirs 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 sub-reservoir is set to be the neighbor of the rightmost cell of the last sub-reservoir, creating a single circular CA. Within each sub-reservoir a random mapping is generated between the elements of the input, of length L_n, and cells of the reservoir. The sub-reservoir size is known as the diffusion length, L_d, where $L_n < L_d$. The random mapping diffuses the inputs into the larger sub-reservoirs.

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 applications with binary input, such the 5-bit memory benchmark, 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 timestep, which is again selectively overwritten according to the input mapping. The rule is applied again, and the process repeats for each timestep 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. The parameters S, N, R, L_d, and I can be set by command line arguments. An alternative scheme for combining the inputs with the reservoir, similar to that adopted by Yilmaz in [16], 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 timestep 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.

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

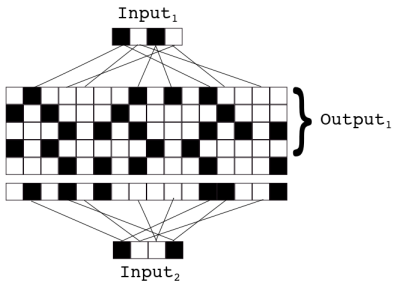

Figure 4: Mapping the input to the reservoir and generating the output. $L_n = 4$, $L_d = 8$, R = 2, I = 4, Rule 90. Initially all the cells of the reservoir are in the same state. At each timestep the input is randomly mapped onto each of the R sub-reservoirs, overwriting the cell state. The CA rule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

3.2. Readout Layer

The 5-bit memory task requires the network to predict three output bits per timestep. Accordingly, the ReCA system that performs it has three output nodes. The weight from identical locations of the CA created at each timestep 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 or greater to 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 scikit functions ran faster, so sci-kit is 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. Stefano Nichele and Magnus Gunderson used SVM in their 2017 ReCA paper on non-uniform reservoirs[13].

4. Benchmark Tests

4.1. Five Bit Memory Task

The 5-bit memory task benchmark tests a network's long-short-term-memory capability, and has been shown to be difficult for recurrent neural networks, including ESN [17][18]. All of the literature on ReCA uses the 5-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 $(i_1+1) mod 2$ (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 timesteps. 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 $o_1 = o_2 = 0$ and $0_3 = 1$. For the remaining five time steps of the run the output bits should be the same as the *message*, i.e. the same as 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 5-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 5-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 5-bit memory task.

4.2. 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 [19], and of a non-uniform ReCA system by Bye [12].

The reservoir continuously evaluates the incoming bits over a window of the last n timesteps, where n is an odd number. The temporal density output node is trained to return 1 if the input window contains more 1s than 0s and return 0 otherwise. The temporal parity output node is trained to return whether the number of 1s in the input window is odd or even. For the temporal density task a delay of τ timesteps 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 timesteps. 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 $n+\tau$ timesteps of the test set. Table 2 shows the system parameters used for all the temporal density and temporal parity results in this paper.

I	**R**	L_d	L_{in}	L_t	τ	**n**
4	12	40	20	400	2	3

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

5. Experiments

5.1. 5-Bit Memory Task

In order to establish a baseline for the performance of the complex CA rules on the 5-bit memory task, the ReCA framework was tested on a selection of elementary CA rules which 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 5-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 times to train and test elementary CA reservoirs of different sizes. 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 [12][10][13].

Finding more effective rules among complex CA 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 5-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 which 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 for elementary CA rules on the 5-bit task.

(I,R)	**Time**
(4,8)	19 s
(4,12)	33 s
(8,8)	57 s

Table 4: Time in seconds to train and test a reservoir using elementary rule 60 on the 5-bit task for different values of I and R. Times are averages of five test runs.

5.1.1. Three State CA 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 5-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 CA 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 5-bit memory task. Those that succeed without any inaccurate predictions are saved for further testing. These are then scored on 100 runs of the 5-bit task. Out of a thousand 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.

5.1.2. Neighborhood Five CA Reservoir

Two state neighborhood five rules have a rule space of $2^32 \approx 4.3 \cdot 10^9$. As with three state neighborhood three rules, this space was searched stochastically, with class I and class II rules ignored. Rules for reservoirs that pass the 5-bit memory task are 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.

5.1.3. 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 were investigated. Those exhibiting class I or class II behavior were rejected while the rest were tested on the 5-bit memory task. Nine rules passed the benchmark. Those that passed were saved and tested on the benchmark 100 times.

5.1.4. Evolving Complex CA Rules

Complex CA 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 are $\approx 3.4 \cdot 10^38$ and $\approx 2.4 \cdot 10^87$ respectively. These spaces are not amenable to a stochastic search for rules able to perform the 5-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 too persist and therefore unable to support computation.

In order to reduce the chaoticity, a genetic algorithm (GA) 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 representation of development in time, 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.

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 are 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 Fig. 5. Each new rule 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.

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 least incorrect predictions when given the 5-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 less than 10 predictions, which was not observed to happen with unevolved rules.

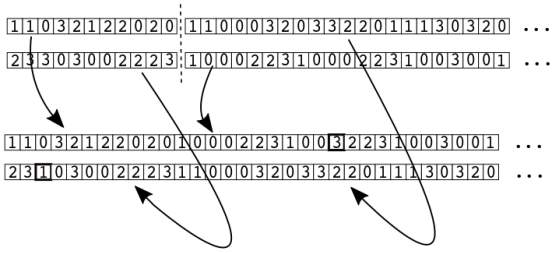

Figure 5: Two rules produce a pair of children by splitting at a randomly chosen point and recombining. Each child receives a single random mutation.

5.1.5. Non-Uniform Reservoir

A non-uniform CA is one in which the cells don't all implement the same rule. The ReCA framework used here implements a parallel non-uniform CA in which the reservoir is split in half into two sub-reservoirs using different rules. The cells at the boundaries between the two sub-reservoirs consider their neighbors in the other sub-reservoir the same as those in their own sub-reservoir in the application of the rule. This allows information to 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 5-bit memory task while other combinations impeded performance[13].

In this work combinations of complex rules were tested together to see whether complementary rule combinations exist for the different types of complex CA rules investigated.

5.2. Temporal Density and Temporal Parity

6. Results

From each of the types of rules that were investigated, the six most successful rules at performing the 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 seen in Table 5.

The selected rules were then tested 100 times each with a range of the reservoir parameters I, R, and L_d. The results can be seen in Table 6.

7. Discussion

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 5-bit memory task, each of the five categories of complex CA rule 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 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 sub-reservoirs from eight to six had a slightly larger effect on the average reservoir performance, which fell to 45.9%.

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
$N5_2$	cddf40
$N5_3$	725fb240
$N5_4$	fd3a6d12
$N5_5$	4f716154
$N5_6$	71ee3aaf
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) = (3,3)	
$S4_1$	b9aa502ddbabb5e8c5b715087a01da08
$S4_2$	e2604a35fca689cefc93a671b4689640
$S4_3$	1f31e800e11c86ba09bafbe00073d533
$S4_4$	bcbb4b3f7b915b47fd154586fa2d15a1
$S4_5$	53bf9dcb97e96e0247c9a980a1291d7e
$S4_6$	73838653ccc875124f5b4658516662c3
Five State Rules (S,N) = (3,3)	
$S5_1$	1871abdbbb1f751c5649ce2208abe0525e5e98543e527293d8fe32c1cbec8f9493da900dc
$S5_2$	159b7bf409f219a835b7bbe69791bfa94432d737117268fff0b99044ac2ccbb513ca91456
$S5_3$	1b4cca207dfd49a79d60a0f0a36fda79a911db434135e22cd79e6e04c69f086c7aab1712a
$S5_4$	1ca6bafcb8cc14e379373cb798461269c4d81258cc28720c74f12dd39b8e48d5863e176a9
$S5_5$	3ac662317b4445a8e63b425645228e27bfc4a0a388497b03a0adede89e87685b12977008f
$S5_6$	53b86edcbce84369a1eeb58079725743b506f228e9459e488139f0763489461f3847c82a

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

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 to complete any successful runs of the task, and the average success rate was only 1.9%.

An attempt was made to find pairs of rules that would complement each other in a non-uniform 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 non-uniform reservoir for any

Rule	(I,R,L_d)=(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	-	-	-	-	-
90	0	-	-	-	-	-
102	9	-	-	-	-	-
153	6	-	-	-	-	-
195	6	-	-	-	-	-
210	6	-	-	-	-	-
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) = (3,3)						
$S4_1$	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) = (3,3)						
$S5_1$	95	50	42	8	15	0
$S5_2$	95	76	70	0	7	0
$S5_3$	98	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 5-bit task for elementary CA and five types of complex CA rule.

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

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)	
$N5_1$	c040f
$N5_2$	ff19e808
$N5_3$	1d5d1f7d
$N5_4$	73ebbf
$N5_5$	fdfc1734
$N5_6$	5d17350f
Four State Rules (S,N) = (3,3)	
$S4_1$	be2614f5ae52e0bceb0564a44a818504
$S4_2$	dd8a5ec44a19cf885d83514504883000
$S4_3$	5c9e5700e4a3423d37456595f56282c0
$S4_4$	ccb14db2d37a6039165a6d45151f4010
$S4_5$	17a2163c27560aaece871545a3208890
$S4_6$	754f9cb2174a865262d53515310f1120
Five State Rules (S,N) = (3,3)	
$S5_1$	94edc5925c6beb872f42363fd4499effd7a06330439d82cd00064a42299880a06a6300f7
$S5_2$	2e9ffae0f04c5a7902a36adca32e1aad8a22e5486eb074b92abcb17ef0e97f22ef3812276
$S5_3$	4a09ab57fa452402cda01dd88d4aa08f24b83f51a5c6a638ba703cbfc0e6d485e7cb98b3f
$S5_4$	4572ec6156d9dab4f6db2cda2bb51a94877dd8d554d71deb9e25698ec08cc0a95ca9ef65
$S5_5$	2c2b5702c45a01eccf1b8f8e29f2c4199f2ea0fa00c9c4e67e2579d1c7f20a770182cca6c
$S5_6$	3e5309a22ea94f23fd18365607c9c64e7201fa39ea33e4b7543332f59c6d43ed51317bbd5

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

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 non-uniform pairings exist which do work well together, it remains an open question how common or rare they are.

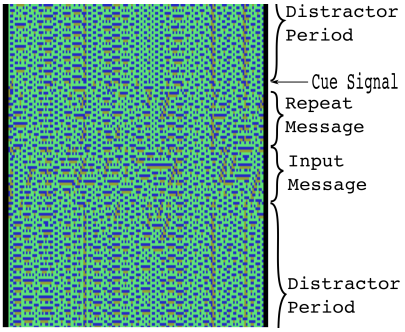

Figure 6: A segment of the history of a reservoir using rule $S3_1$ on the 5-bit task with I=4, R=8, and D_L=30. Each four horizontal lines is one reservoir timestep. At the top of the image is the end of the distractor period after the input message 00001. The reservoir has fallen into a stable repeating 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 00010 is input the reservoir quickly falls into another stable pattern that encodes the new message.

8. Conclusion

A reservoir computing with cellular automata (ReCA) framework has been implemented which expands the field of ReCA research from the 256 elementary cellular automata rules to investigate rules with larger neighborhoods and more states, here called *complex* CA rules. A genetic algorithm 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 5-bit memory task benchmark.

The six best rules from each of five categories of complex rule 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.

References

[1] H. Jaeger, "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*, vol. 148, no. 34, p. 13, 2001.

[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*, vol. 14, no. 11, pp. 2531–2560, 2002.

[3] M. Lukoševicius and H. Jaeger, "Overview of reservoir recipes," tech. rep., School Eng. Sci., Jacobs Univ, Bremen, Germany, 2007.

[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," *arXiv preprint arXiv:1808.04962*, 2018.

[5] C. G. Langton, "Computation at the edge of chaos: phase transitions and emergent computation," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1-3, pp. 12–37, 1990.

[6] R. Legenstein and W. Maass, "Edge of chaos and prediction of computational performance for neural circuit models," *Neural Networks*, vol. 20, no. 3, pp. 323–334, 2007.

[7] S. Wolfram, *A new kind of science*, vol. 5. Wolfram media Champaign, IL, 2002.

[8] A. Wuensche, "Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the z parameter," *Complexity*, vol. 4, no. 3, pp. 47–66, 1999.

[9] M. Cook, "Universality in elementary cellular automata," *Complex systems*, vol. 15, no. 1, pp. 1–40, 2004.

[10] O. Yilmaz, "Reservoir computing using cellular automata," *arXiv preprint arXiv:1410.0162*, 2014.

[11] M. Margem and O. Yilmaz, "An experimental study on cellular automata reservoir in pathological sequence learning tasks," 2017.

[12] E. T. Bye, "Investigation of elementary cellular automata for reservoir computing," Master's thesis, NTNU, 2016.

[13] S. Nichele and M. S. Gundersen, "Reservoir computing using non-uniform binary cellular automata," *arXiv preprint arXiv:1702.03812*, 2017.

[14]

[15] S. Nichele and A. Molund, "Deep reservoir computing using cellular automata," *arXiv preprint arXiv:1703.02806*, 2017.

[16] O. Yilmaz, "Connectionist-symbolic machine intelligence using cellular automata based reservoir-hyperdimensional computing," *arXiv preprint arXiv:1503.00851*, 2015.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] H. Jaeger, "Long short-term memory in echo state networks: Details of a simulation study," tech. rep., Jacobs University Bremen, 2012.

[19] D. Snyder, A. Goudarzi, and C. Teuscher, "Computational capabilities of random automata networks for reservoir computing," *Physical Review E*, vol. 87, no. 4, p. 042808, 2013.