The Dynamical Landscape of Reservoir Computing with Elementary Cellular Automata

Tom Eivind Glover[1,*], Pedro Lind[1], Anis Yazidi[1], Evgeny Osipov[2], and Stefano Nichele[1,3]

[1]AI Lab, Department of Computer Science, Oslo Metropolitan University, Oslo, Norway
[2]Department of Computer Science Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden
[3]Department of Holistic Systems, Simula Metropolitan Centre for Digital Engineering, Oslo, Norway
[*] Corresponding author: tomglove@oslomet.no

Abstract

Reservoir Computing with Cellular Automata (ReCA) is a promising concept by virtue of its potential for efficient hardware implementation and theoretical understanding of Cellular Auotmata (CA). However, ReCA has so far only been studied in exploratory studies. In this work, we take a more in depth view of the landscape of Elementary Cellular Automata for Reservoir Computing. In this paper, the ReCA is applied to the X-bit memory benchmark with a thorough exploration for key parameters including number of random mappings (R), number of bits (N_b) and size of the vector that the random mapping is mapped to (L_d). Our evidence shows that the parameter space, including the full panoply of CA rules, is much richer then what previous evidence indicates. This suggests that some CA rules would require careful consideration and custom parameters setup.

Introduction

Modern Machine Learning (ML) has given great benefits to society, but it comes at a cost. One such cost is the energy requirement to train modern ML algorithms, such as Deep Learning (DL) with error backpropagation (Strubell et al., 2019). Top-down approaches, such as DL, require careful design and engineering of the network architectures to solve the given task, as well as global error calculations used for training, which implies a form of centralized control.

Reservoir computing (RC) is a bottom-up approach which shows potential to reduce the training time. This is performed simply by projecting the input through a non-trained, but dynamically set-up substrate and separating the high-dimensional results through a linear readout layer. The concept originated in Echo-State Network (ESN) using a Recurrent Neural Network (RNN) as a substrate (Jaeger, 2001) and in Liquid State Machines (LSM) where a Spiking Neural Network served as a substrate (Maass et al., 2002). Since then, both concepts have been grouped under the umbrella term of RC, and other substrates have been explored. One such substrate is Cellular Automata (CA).

CA can be considered to be a special case of (neural) networks, where connectivity is ordered such that every node is connected to exactly the same number of neighbours, which in turn are uniformly connected to each-other. One advantage of CA is that they can be implemented in hardware, such as in Field Programmable Gate Arrays (FPGA) (Morán et al., 2019), which results in energy efficient computing substrates as well as fast execution due to the hardware supporting CA implementation natively and CA being fully parallel.

Reservoir computing with Cellular Automata (ReCA) is the version of RC using CA as the substrate for the reservoir. The concept was first explored in Yilmaz (2014). Since then, ReCA has been studied in several exploratory studies (Nichele and Molund, 2017; Nichele and Gundersen, 2017; Margem and Gedik, 2019; Kleyko et al., 2020; McDonald, 2017; Babson and Teuscher, 2019). All of these studies utilize CA as part of the reservoir, but none of them explore the entire rule-space of Elementary Cellular Automata (ECA). By accumulating some of the findings from these studies the reservoir can be made much smaller then the original studies, such that exploring the entire rule-space becomes more computationally viable. This is beneficial in order to gain a better understanding of the important factors that need to be considered when practically setting-up a RC system.

We introduce some of the relevant concepts this paper builds on in the upcoming background section. The third section introduces the methodology of the 5 bit benchmark as well as the ReCA implementation, and two performance metrics, namely perfect score and weighted average. The fourth section introduces the experimental setup where the extended information is given in order to replicate this study. The fifth section documents the conducted experimental results and their analysis. The sixth section includes the discussion and conclusion where the key findings are shown and discussed. Finally, the future work section is presented.

Background

Cellular Automata

In this work, ECA is used as a computing substrate. ECA is a special case of CA, where each cell has 2 possible states and 3 neighbours, where 1 neighbour is the cell itself. A common way to handle how to update the CA to the next state is a Transition Table (TT). This method relies on using a lookup-table where the current state of the relevant neighbourhood is used to look up what the next state of the cell is. Due to the limits on the number of neighbours and states, ECA has 256 unique rules in the rule-space. The index of ECA's TT is fixed, therefore the rules can be compressed to just the results which is in essence a 8-bit binary string. The conversion of the rules binary string in decimal is used as convention to name the rule, e.g., $bin(01011010) = dec(90)$.

ECA have been extensively studied and classified most notably by Wolfram (Wolfram, 2002). CA rules have been grouped into 4 classes of behavior, i.e., uniform (Class 1), periodic (Class 2), chaotic (Class 3) and complex (Class 4). In Wolfram (1986), many of the rules were found to be equivalent to other rules, this reduces the rule-space to 88 unique rule groups: each group, having a Minimum Equivalence (ME) rule that represents its rule group. A simple overview of the Equivalence classes and associated Wolfram classification can be found in Martinez (2013).

ECA with Memory (ECAM)

ECAM is a classification of ECA based on how the ECA rule changes behaviour in regard to the Wolfram classes used in combination with a memory rule (Martinez et al., 2013). In contrast to ECA, ECAM makes use of more then just the previous state of the CA. This is performed by either applying the function majority, minority or parity to the x previous steps and then using the same ECA TT to select the next state. The ECA rule-space fits into 3 different classes in regards to how they changed behavior.

Strong: Most memory functions change the rule to another different class quickly

Moderate: Memory functions can transform to a different class and conserve the same class as well

Weak: Memory functions are unable to transform into another class

One might notice how this definition is different from the form presented in Martinez et al. (2013). There appears to be an inaccuracy in the definition of the weak and strong, which are flipped. As the definition contradicts the propositions and evidence in the paper, a correction would be in order. Therefore, the definitions have been changed to account for this. In fairness, one could also have flipped which rules are strong and weak, instead of flipping the definitions.

Edge of Chaos and λ Parameter

"Edge of chaos" is a term used to name the region in the parameter space of complex systems displaying transition between ordered and disordered behavior. Here it is theorized that computation defined as transformation, manipulation and storage of information is found. In Langton (1990), it was shown that the rule-space of 1 dimensional multi-state enlarged neighbourhood CA forms a phase transition between order and chaos when organized over a parameter, named λ (Lambda).

Reservoir Computing

In simple terms, RC consists of 3 parts, the input layer, the untrained reservoir layer, and the (linear) output layer. The input layer maps the input into the untrained reservoir layer, the reservoir layer projects the input into higher dimensions, and the output layer reads the reservoir state and extract the useful features. Since its origin in ESN (Jaeger, 2001) and LSM (Maass et al., 2002), RC has expanded into many different substrates. Some alter the structure, e.g., in Gallicchio and Micheli (2016) a strategy using a deep layered structure of several reservoirs was employed. Other change the reservoir substrate, e.g., in Jensen and Tufte (2020) the substrate is Artificial Spin Ice which consists of a simulated lattice of nanomagnets. RC has also been implemented in many physical systems (Tanaka et al., 2019). One novel concept is implementing the RC framework using Biological Neural Networks (BNN), by growing neurons on a Micro Electrode Array (MEA) (Aaser et al., 2017). The BNN can be stimulated using electric signal and the MEA can also read the state of a region of neurons. It has also been demonstrated that the RC framework can work on a bucket of water, encoding the input using motors as waves on the water surface that is recorded with a camera. The recording is then used by a simple perception to solve XOR and speech recognition problems (Fernando and Sojakka, 2003).

Reservoir Computing with CA (ReCA)

The first study that introduced CA as a substrate in reservoir computing is Yilmaz (2014). In this study, Game of Life and several ECA rules were investigated as reservoir substrates and tested on a 5 bit and 20 bit memory benchmark (adapted for CA). In addition it presents, a theoretical comparison of CA vs ESN, using the metric of number of operations needed to solve the benchmark, which documents a clear advantage of using CA.

Since then, other works have studied ReCA using the 5 bit memory benchmark. In Nichele and Molund (2017), the structure of the CA was changed to a deep layered architecture and compared to a single layer, which resulted in noticeable improvements of performance. In Nichele and Gundersen (2017), the CA substrate was organized as consisting of two regions of different ECA rules. Different combinations of rules were explored, some of them showed great promise. In Margem and Gedik (2019), different methods of cell history selection that are used for the classification model are explored on the 5 bit memory task and in addition a temporal order task and arithmetic and logic operation tasks. In Babson and Teuscher (2019), CA rules with multiple states and larger neighbourhoods were evolved and then tested on the 5 bit memory benchmark.

ReCA is also used on other benchmarks then the 5-bit memory benchmark. E.g. in Morán et al. (2019), ReCA is implemented in hardware and tested using MNIST.

Methodology

X-bit Memory Task

If one considers the reservoir substrate and classification model as a black box, then it is evident that the X-bit memory task is one of the simplest IO binary classification task with a temporal element.

Step	Input				Output			Stage
1	1	0	0	0	0	0	1	Input bits
2	1	0	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
7...203	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	Output bits
207	0	0	1	0	1	0	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: Example of the 5 bit memory task with distractor period of 200 and Input of the number 25 in binary form. Artifact inspired by Babson and Teuscher (2019)

The X bit memory benchmark is designed to test long-term-short-term memory capability of a reservoir under perturbation. To do so, it is designed in 4 stages seen in Table 1. The first stage encodes the input into the substrate, the second stage inputs noise into the substrate, the third stage sends the cue signal to tell the model to start returning the binary input. The final 4th stage is the output stage when the originally input bits are to be returned. The benchmark has 4 input channels consisting of the actual bits to input, the flipped bits to input, the distractor channel and the cue signal channel. The task has 3 output channels: one channel to return the input, a second channel to return the flipped input and a final channel to continually report that the system is not in the binary output stage (or simply put 1s until when the cue signal is given). As one and only one channel returns 1 at each time, this can be considered a single label classification task.

ReCA X-Bit Memory Task

Now that the benchmark has been presented independently of the chosen substrate, we detail the specific benchmark definition in the context of a CA substrate.

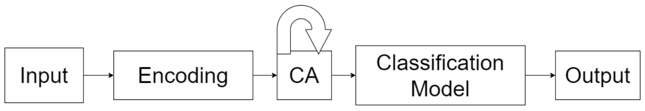

Figure 1: Simple model of the different parts of ReCA using the X-Bit memory benchmark.

As shown in Figure 1, some steps have been added. The Encoding part comes first which considers how to inject the input into the substrate. For binary data there has been many different methods proposed and or implemented (Yilmaz, 2014; Nichele and Molund, 2017; Margem and Gedik, 2019), but there are some common tendencies. The input is usually randomly mapped into the CA in several redundant mappings. This random mapping stays fixed throughout the experiment and is usually done multiple times. The number of times is given by the redundancy (R) parameter in the experiment. In Yilmaz (2014), this was done using a vector of the same size as the input, but in later experiments (Nichele and Molund, 2017; Nichele and Gundersen, 2017) a larger vector is used. The size of this new vector is given by the L_d parameter. The simplest way to encode the input into the CA is to overwrite the current state of a mapping. Since this might overwrite important information in the CA, other methods have been used, such as simple binary operations between the current state of the CA and the new input value.

The second component is the actual substrate itself, a CA in the work herein. This operates as any CA, and can be of any dimensionality, with any number of states, and with any neighbourhood scheme. Commonly, the CA permutes the input a number of steps before given the next input. Therefore, one can say that the CA and X-Bit memory benchmark exist on different timescales. The number of steps before the next input is represented by I, which is another parameter in the experimental setup.

The third part is the classification model. This stage uses the state of the CA (or a set of previous states) to classify and produce an output state. Due to the nature of RC, the classification model is commonly a linear model. Consider if one were to use a Deep Neural Network as a classification model rather then a linear one, this would cast doubt about whether the separation was done in the model or the substrate. Therefore using a linear model demonstrates that the system is actually doing RC. When limited by this requirement, there are two common models used, i.e., linear regression and Support Vector Machine (SVM) with a linear kernel. If linear regression is used, one would need to pair it with a rounding function, or some max confidence method in order to produce a clear output. If SVM is used, one takes advantage of the fact that the output is always only a single class and can therefore handle the model as a standard SVM model.

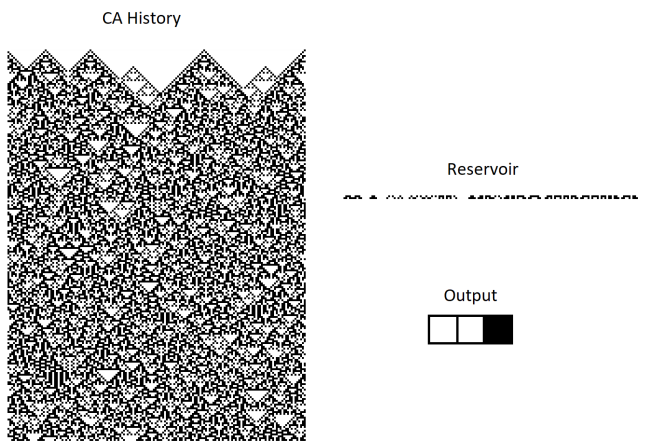

Figure 2: Snappshot of ReCA with $N_b = 5, R = 4, I = 2, L_d = 40, D_p = 60$ and Rule 90

Perfect Score Metric vs Weighted Average Metric

The common way to score the ReCA system on the X-bit memory task is to run through all permutations of the X-bits. The example in Table 1 represents only one out of 32 possible input permutations. In order to get a perfect run, not only all permutations would need to be correct, but every step of all the permutations needs to be correctly classified. For a single run, that means $210 * 32 = 6720$ correctly classified states and a single wrongly classified state result in a 0 score on the full run. This method is strict, and it leaves little room to identify setups that are close to solving the problem. We introduce here a strategy based on a Weighted Average ($\bar{W}$) metric to address this. Consider Table 1, one might notice that there is one output that is more common then the others. This "no output state" is so common that in the example, $W = 205/210 \approx 0,976$ of the classifications are the same. This region can be easily guessed correctly, so all the difficulty of this task lies in the final x-bits. In order to better inspect the results, one can stretch that region by applying a $\bar{W}$ metric. Given that the length of the distractor period is D_p, the number of bits is N_b and the fraction of correctly classified states is $\bar{S}_c$, the following formula applies:

$$W = \frac{D_p + (N_b)}{D_p + (2 * N_b)}, \qquad \bar{W} = \frac{\bar{S}_c - W}{1 - W}.$$

If a run got $\frac{6700}{6720} \approx 0.997$ on average correctly classified states, its $\bar{W}$ would therefore be $\frac{0.997-0,976}{1-0,976} = 0.875$

Experimental Setup

Libraries, frameworks and Source Code

In order to carry out the experiments efficiently, the EvoDynamics framework was used to build the CA substrate. EvoDynamics by Pontes-Filho et al. (2020) natively supports CA reservoirs and is built upon TensorFlow, and therefore can make use of TensorFlows-GPU optimised code. This still does not reach the full potential of optimization for CA. As an example, it still relies on float point, which is strictly not necessary when working with CA, but EvoDynamics strikes a good balance between performance and implementation time. In addition SciKit-learn was used to create and train the SVM model (Pedregosa et al., 2011) that is used as the classification model in this work.

The source code and extended results for this paper can be found at the DeepCA project github[1].

Exploring the Parameter Space

The full parameter space of the framework comprehends six parameters: the redundancy R, the number of bits N_b, the size of the vector the input is mapped to L_d, the CA permutation iteration and reservoir height I, the distractor period D_p and the CA rules.

In this paper, the parameter of R is selected for exploration due to performance reliably increasing over increasing R in past experiments (Yilmaz, 2014; Nichele and Molund, 2017; Nichele and Gundersen, 2017). N_b was selected because it was the most promising candidate for making the problem difficulty harder or easier depending on value. L_d is explored due to the surprising results of the R parameter experiment. As this paper only explores 3 parameter, the $D_p = 200$ and $I = 2$ have remained fixed for all experiments described in this paper. They remain to be explored in future experiments. To be precise, as the experiment exists on two different timescales, i.e., the benchmark and the CA timescale, $D_p = 200$ is on the benchmark timescale (not the CA). This means that 200 perturbations have been done to the CA. The $I = 2$ parameter affects the experiment in two ways. Firstly, it is directly linked with how many steps of the CA is fed into SVM. $I = 2$ means that the last two states of the entire CA width is used to classify an output. Secondly, it is linked to how many CA development steps are used before another input is given to the system. This paper uses the input step also as a CA development step, meaning for every input step on the timescale of the benchmark, 3 CA steps are performed. An argument for why I is linked to two different concepts in the experiment, is that if one were to predict from the classification model on more previous states then available between two inputs, the classification model would be able to access the previous input directly, rather than constructing it from the afterimage.

By fixing the two parameters above, we focus henceforth in the region of the parameter space defined by R, N_b and L_d. By exploring the behavior of CA rules in relation with the variation of these three parameters, one is able to assess important features of CA reservoirs. In order to evaluate

[1]https://git.io/JqCfW

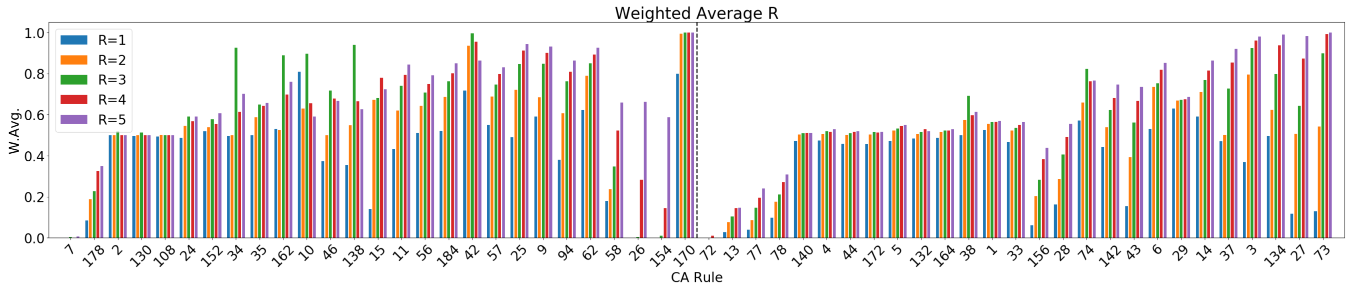

Figure 3: Strong and moderate class 2, using the $\bar{W}$ metric and $R = \{1, 2, 3, 4, 5\}$

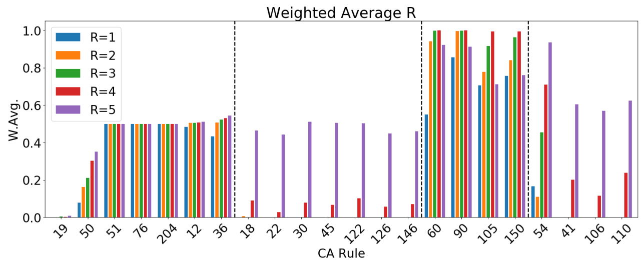

Figure 4: Weak class 2 and all class 3 and class 4, using the $\bar{W}$ and $R = \{1, 2, 3, 4, 5\}$

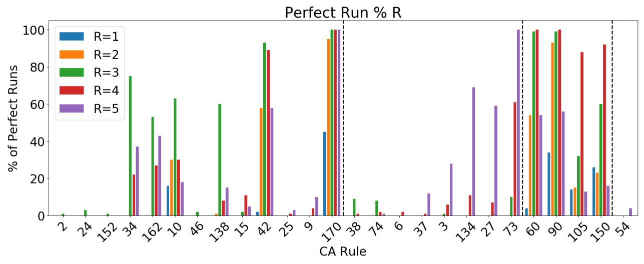

Figure 5: Perfect runs metric in % and $R = \{1, 2, 3, 4, 5\}$

memory, by varying R it is possible to identify which rules are affected by redundancy. By varying N_b, a threshold for memory can potentially be identified. By varying L_d the grid-sizes dynamical relation to memory can be identified.

If not otherwise stated $R = 4$, $N_b = 5$, $L_d = 40$ and ECA rules are always varied in all experiments. Using these values, the full width of the CA can be calculated by $R * L_D = 4 * 40 = 160$. The number of states given to the SVM is calculated by $R * L_D * I = 4 * 40 * 2 = 320$

Encoding Strategy

As previously mentioned, different encoding strategies have previously been employed. This experiment uses XOR to encode the current step of the CA with the input. This means that the state of the CA is always modified if the input is 1.

Results: from Parameter Space to Memory Features of CA Behavior

In this section, the results from the experiments are presented. First, the experiment where the parameter of R is shown, followed by the experiment on the parameter N_b and finally L_d. For each experiment, we will review and discuss the results of the $\bar{W}$ metric first, followed by the Perfect run metric. The two main experiments parameters for N_b and R were conducted over the entire Rule-space of ECA, but due to limited space the only rules that are presented are the Minimum Equivalence (ME) rules. Furthermore, rules that only achieved 0.005 $\bar{W}$ or less score on all task were removed. In all the experiments rule 0, 8, 23, 32, 40, 104, 128, 136, 160, 168, 200, 232 did not achieve this and therefore are not present in any figure. Note that this list includes only rules that are classified in Wolfram class 1 or 2, and that this list includes all Class 1. This is explained by the fact that as Class 1 rules quickly transform into uniform final states, any input would be quickly forgotten.

The results were sorted using Wolframs classes, then by ECAM category, then finally by $\bar{W}$ Performance on the 3 bit memory, in sinking order of importance. This was performed to highlight association between ECAM and Wolfram classes and behaviour within this experiment. On the $\bar{W}$ the results are also split between Strong and Moderate Class 2, and the rest in order to better fit the width of this paper. All subgroups are separated by a dashed line. All results given are averages over 100 runs and the SVM is fully trained on all permutations of the input.

The Impact of Redundancy R on CA Behavior

The first results presented here are for the experiments where R was the dependent parameter over the $\bar{W}$ performance, as seen in Figure 3 and Figure 4. Looking at the macro perspective, one can already notice several different behaviors. By varying the R, some rules behave stably, this meaning that increasing or decreasing R does not affect performance. Some have an increase in performance with increase in R. While some are more vulnerable to sufficient R to achieve any score at all, and some seem very dynamic over R and surprisingly show that more R is in fact not better.

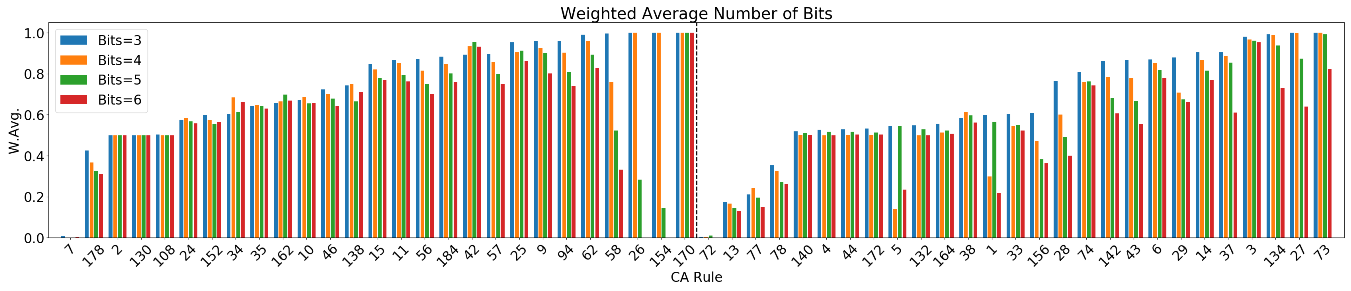

Figure 6: Strong and moderate class 2, using the $\bar{W}$ metric and $N_b = \{3, 4, 5, 6\}$

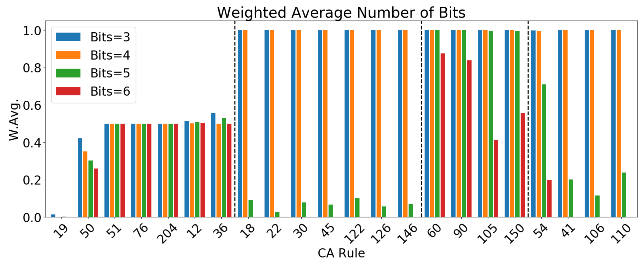

Figure 7: Weak class 2 and all class 3 and class 4, using the $\bar{W}$ metric and $N_b = \{3, 4, 5, 6\}$

Some of the rules that are stable over R actually highlight a weakness in the $\bar{W}$ scoring method. If one considers this benchmark, the worst scoring rules do get 0 on this scoring method, but rules that can never get a perfect score are able to get 50% reliably as long as they detect the cue signal. One needs only to consider rule 204 to understand this. Rule 204 always projects the current state as the next step, meaning that the binary input will be removed again with the next 1 from the input. This rule is incapable of separating the input but it is very capable of detecting the cue signal, as it has its own input channel. This means that rule 204 can, under any rational condition, get 50% score but never more. Many rules in class 2 seem to rely on this trick.

Some rules are stably increasing over R to points beyond 50%, meaning they must be able to remember some of the input bits. Some rules only get any performance at all if they have sufficient R, all rules that are strong class 3 behave this way, and so do most of the class 4, with a slight exception for rule 54. The most surprising results are the rules that perform worse with increased R. Many of the strong and moderate class 2 rules have peaks at $R = 3$ and the weak class 3 has a peak at $R = 4$. Considering this, it is likely that other rules that have not yet received a sufficient R might also have peaks, only outside the scope of this experiment.

Let us consider the Figure 5 where the perfect run in % metric is shown on the R parameter experiment. High performance is more sparse when using this metric, yet one can still see most of the same set of behaviors. Some rules, such as rule 73, stably increase over R. While some rules peak at $R = 3$ or $R = 4$, mostly in the same associated manner to ECAM as using the previous metric.

Another surprising results is, if one considers both metrics, there are some rules that can remember many permutations of input but never manage to achieve a perfect result, such as in rule 62, which gets reliably a very good score on the $\bar{W}$, but no score on perfect runs.

Some additional rules not identified in previous literature can also be highlighted here, Rule 42 and 170. The X-Bit memory benchmark only features memory and rule 170 projects sideways, so as long as the CA is not saturated, the trajectory of the input can easily be backtracked. Rule 42 employs largely the same strategy, in fact its TT is only different by a single bit, therefore it would only act differently in only in a single neighbourhood scheme. Considering that this benchmark tests memory, in a very different benchmark which relies on complex interaction between the input these rules could perform poorly. No weak class 2 or strong class 3 are able to solve a perfect run at all. The rules previously identified as high performers on this benchmark still perform well. These being the Rules 60, 90, 105 and 150, which are all weak Class 3. Rule 60, 90 and 150 are all Additive Cellular Automata (ACA) which rely on more then a single neighbour. More details on ACA can be found at (Rowland and Weisstein, 2021). Rule 105 is a compliment of rule 150 but is not not normally considered equivalent.

The Impact of N_b on CA Behavior

In Figure 6 and Figure 7 we explore the parameter of N_b on the $\bar{W}$ metric. It can be seen that most of the class 2 have comparatively little variation in performance of a 3 bit compared to a 6 bit. This is in contrast to class 3 and 4 rules, and especially true for strong Class 3 and most class 4 (Rule 41, 106 and 110) which on the $\bar{W}$ $N_b = 6$ gain 0 score and on the $N_b = 4$ nearly perfect score.

The behavior of rule 5 and 1 was surprising, as they seem to find $N_b = 5$ easier to memorize then 4. One can speculate that this might be due to sensitivity of the exact execution length of the CA between the input is projected into the reservoir and when it is required to give it out, as the

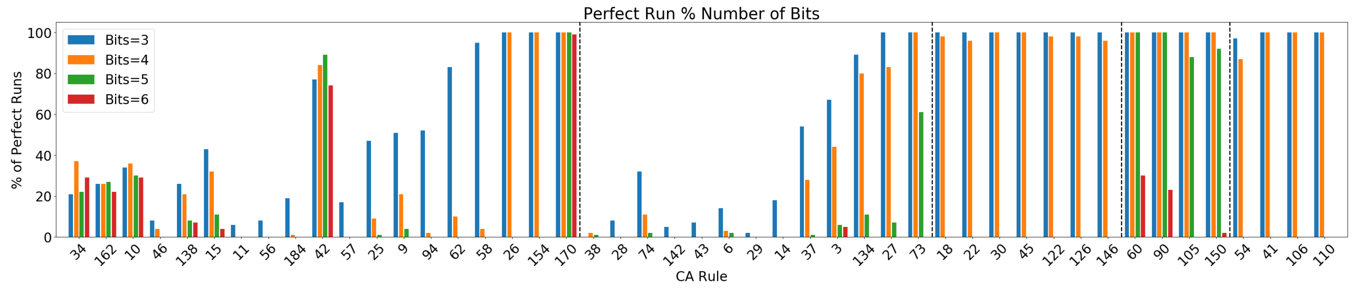

Figure 8: Perfect run metric % and $N_b = \{3, 4, 5, 6\}$

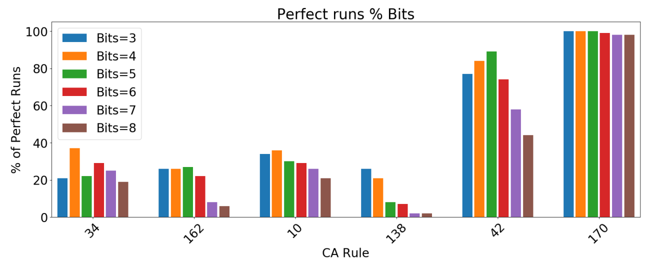

Figure 9: Additional 7 and 8 N_b for stable rules.

total length is increased by $(I + 1) = 3$ CA steps per bit. Therefore, the result may be dependant on an exact length.

When one considers Figure 8 where we explore the parameter of N_b on the perfect score in % metric, there are still rules (marked by their absence) that are incapable of getting any perfect run even on $N_b = 3$. In fact, no weak class 2 is able at all, but considering that most of them seem to only ever be able to remember the cue signal but not the input bits, this is probably the cause. Also considering weak class 2 are the rules that continue to have class 2 behaviour when any memory function is applied it makes sense that these rules that only project downwards fit into this category. All class 3 and 4 move quickly between finding the task trivial and impossible (particularly true for strong class 3 and class 4).

Considering these two experiments, some rules behave out of place. Rule 26 and 154 behave as strong class 3 and class 4, and class 4 minus rule 54 behave like the strong class 3 and visa versa. If one considers that complexity is found at the edge of chaos, and that some orderly rules from class 2 and chaotic rules from class 3 are indistinguishable from some of the complex class 4, this is not unimaginable.

Some of the rules seem to in contrast to many others, find very little variation in how difficult $N_b = 3$ and $N_b = 6$ are to memorize. If one extends this into $N_b = 7$ and then $N_b = 8$ only strong class 2 manage to do this. The performance on the perfect run in % metric can be seen in Figure 9. These rules largely seem to not find 8 bit more difficult then $N_b = 3$, considering that $N_b = 8$ has 256 different permutations and $N_b = 3$ only 8, this is quite remarkable. Considering also that many of these rules do not reliably solve the problem, they must be very dependent on the variability of the experiment, which is the random mapping set at initialization. It is hypothesized that these rules have a variation of beneficial and detrimental mappings. This could potentially explain why they have peaks at $R = 3$, if this is the point where the likelihood of getting a detrimental mapping outweighs the likelihood of getting a beneficial mapping. Rules with peak at $R = 4$ could simply have a different distribution of beneficial and detrimental mappings.

In the experiments presented earlier, only results for the ME rules of ECA have been shown. In terms of variation in range of behavior the ME rules are sufficient, however some rules considered to be equivalent achieve very different results in this experiment. For example, Rule 18 and rule 183 are considered equivalent in Wolfram (1986) and Wuensche et al. (1992), but perform rather differently. The same is observed for rules 200 and 236.

The Impact of L_d and Grid-size on Performance

We saw from the earlier results that some rules were very dynamic based on their configuration, and that some rules had peaks of $R = 3$ or $R = 4$ (in the parameter space explored). One could assume that the SVM used is being overloaded with the very large dimension of the data it is classifying, but as the peaks are in different locations and some rules seem to still have an increasing performance, this is not a sufficient explanation on its own. Furthermore previous experiments have used even larger dimensional data for SVM classification and still report success (Nichele and Gundersen, 2017). These peaks are in somewhat of a contrast to what is previously discovered (Yilmaz, 2014; Nichele and Molund, 2017; Nichele and Gundersen, 2017), where higher R seems better, but these previous experiments do not actually report odd number values.

Consider that R does not only affect the level of redundancy in the mapping but also the total width of the CA. Could the parameter space of L_d be very dynamic? In this experiment presented here, the goal is to remove R from the experiment by setting it as 1, and explore some rules which

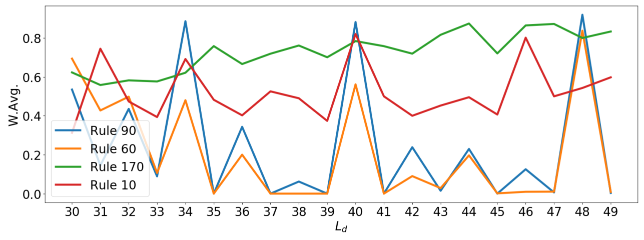

Figure 10: Parameter space of Rule 90, 60, 170 and 10, using the $\bar{W}$ metric, $R = 1$ and $L_d = \{30, 31, ..., 48, 49\}$

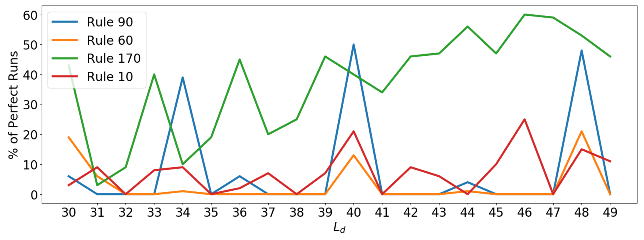

Figure 11: Parameter space of Rule 90, 60, 170 and 10, using the perfect runs metric in %, $R = 1$ and $L_d = \{30, 31, ..., 48, 49\}$

are still able solve the benchmark when $R = 1$. Note, that when $R = 1$ the only variable that affects grid-size is the L_d variable. Stable rules are also contrasted with more dynamic rules. Figure 10 and Figure 11 explore several rules. Rule 170 which is very stable and in general by a decent margin the best performing rule, rule 10 which shows a peak at $R = 3$, but behaves largely stable with different N_b, and rule 60 and 90 which show a peak at $R = 4$ and vulnerable to increase in N_b. The parameter space of these 4 groups show more dynamical behavior then expected. Rule 60 and 90 show very dynamical behavior where just the small change between $L_d = 40$ and $L_d = 39$ goes from one of the best performing configuration to one of the worst. A strong trend in Rule 60 and 90 for high performance on even number and poorly with odd numbers can be observed. Rule 10 shows some dynamic behavior as well, but not as much as Rule 60 and 90. Rule 170, has the most stable performance and shows an increasing trend upwards over increasing L_d, but there is also some dynamic behavior in particular in the lower region of L_d.

Discussion & Conclusion

Categories

In the different parameter spaces explored in this work, some different behaviors is shown. The behavior classified as incapable represents the rules that are not able to memorize a single bit. The rules classified as cheaters are able to detect the cue signal and guess about 50% of the time. They are only reliable to memorize a single bit. Rules fall within a range of stable to vulnerable when mappings are initialized randomly. Some rules are stable with regard to parameter setup, but very dependent on the different random initialization. This poses the question of what is a beneficial mapping. Rules fall within a range of stable to vulnerable when regards to the parameter configuration. Some rules are quickly changing between finding problems trivial and impossible based on a small changes in the parameter configuration. This indicates that the parameter space for these rules is very dynamic and needs care when selecting values.

CA behavior association with ECAM We have seen that some of the behaviors are strongly associated with ECAM classifications and the Wolfram classification. Considering that ECAM classifies ECA by behavior over different memory functions and the x-bit memory task evaluates memory capabilities, therefore, some association is expected.

High sensitivity to CA total width We have shown that some rules are very dynamical and sensitive to CA width configuration. Grid size was also found to be important in rule 90 (Kleyko et al., 2020), but in contrast, odd numbers, and especially primary numbers were perceived to be beneficial, but on the metric of the rules randomization period.

Performance of ACA We see that many of the ACA rules perform well and that rule 170, performs better on the X-Bit Memory benchmark then the rules previously highlighted (rule 90). Considering that rule 170 is only using a single neighbour cell to define its state, its hypothesized that on a benchmark featuring complex interaction rather then just memory performance would be different.

Future Work

In this work, we have shown that the regions of the parameter space are particularly dynamic (within the regions explored). Only some of the defined parameters for this benchmark have been examined systematically. Parameters such as D_p and I remain to be explored further. Some rules are hypothesized to have a variation of beneficial and detrimental mappings. Therefore, a promising direction of exploration is to better predict robust mappings in order to identify beneficial configuration for ReCA more precisely. The X-Bit Memory task only tests memory capability. It would be beneficial to conduct similar studies to this paper on other benchmarks. Specifically ones that require transformation and manipulation of the input.

Acknowledgements

This work was partially financed by the Research Council of Norway's DeepCA project, grant agreement 286558.

References

Aaser, P., Knudsen, M., Ramstad, O. H., van de Wijdeven, R., Nichele, S., Sandvig, I., Tufte, G., Stefan Bauer, U., Halaas, Ø., Hendseth, S., et al. (2017). Towards making a cyborg: A closed-loop reservoir-neuro system. In *Artificial Life Conference Proceedings 14*, pages 430–437. MIT Press.

Babson, N. and Teuscher, C. (2019). Reservoir computing with complex cellular automata. *Complex Systems*, 28(4):433–455.

Fernando, C. and Sojakka, S. (2003). Pattern recognition in a bucket. In *European conference on artificial life*, pages 588–597. Springer.

Gallicchio, C. and Micheli, A. (2016). Deep reservoir computing: A critical analysis. In *ESANN*.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13.

Jensen, J. H. and Tufte, G. (2020). Reservoir computing in artificial spin ice. In *Artificial Life Conference Proceedings*, pages 376–383. MIT Press.

Kleyko, D., Frady, E. P., and Sommer, F. T. (2020). Cellular automata can reduce memory requirements of collective-state computing. *arXiv preprint arXiv:2010.03585*.

Langton, C. G. (1990). Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1-3):12–37.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

Margem, M. and Gedik, O. S. (2019). Reservoir computing based on cellular automata (reca) in sequence learning. *Journal of Cellular Automata*, 14.

Martinez, G. J. (2013). A note on elementary cellular automata classification. *arXiv preprint arXiv:1306.5577*.

Martinez, G. J., Adamatzky, A., and Alonso-Sanz, R. (2013). Designing complex dynamics in cellular automata with memory. *International Journal of Bifurcation and Chaos*, 23(10):1330035.

McDonald, N. (2017). Reservoir computing & extreme learning machines using pairs of cellular automata rules. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2429–2436. IEEE.

Morán, A., Frasser, C. F., Roca, M., and Rosselló, J. L. (2019). Energy-efficient pattern recognition hardware with elementary cellular automata. *IEEE Transactions on Computers*, 69(3):392–401.

Nichele, S. and Gundersen, M. S. (2017). Reservoir computing using non-uniform binary cellular automata. *arXiv preprint arXiv:1702.03812*.

Nichele, S. and Molund, A. (2017). Deep learning with cellular automaton-based reservoir computing. *Complex Systems*, 26(4):319–339.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Pontes-Filho, S., Lind, P., Yazidi, A., Zhang, J., Hammer, H., Mello, G. B., Sandvig, I., Tufte, G., and Nichele, S. (2020). A neuro-inspired general framework for the evolution of stochastic dynamical systems: Cellular automata, random boolean networks and echo state networks towards criticality. *Cognitive Neurodynamics*, pages 1–18.

Rowland, T. and Weisstein, E. W. (2021). Additive cellular automaton. `https://mathworld.wolfram.com/AdditiveCellularAutomaton.html`, Online; accessed 03.05.2021.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123.

Wolfram, S. (1986). Theory and applications of cellular automata. *World Scientific*.

Wolfram, S. (2002). *A new kind of science*, volume 5. Wolfram media Champaign, IL.

Wuensche, A., Lesser, M., and Lesser, M. J. (1992). *Global Dynamics of Cellular Automata: An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata*, volume 1. Andrew Wuensche.

Yilmaz, O. (2014). Reservoir computing using cellular automata. *arXiv preprint arXiv:1410.0162*.