NTNU
Norwegian University of
Science and Technology

Reservoir Computing Using Quasi-Uniform Cellular Automata

Magnus Skogstrøm Gundersen

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Gunnar Tufte, IDI
Co-supervisor: Stefano Nichele, IDI

Norwegian University of Science and Technology
Department of Computer Science

Summary

Unconventional computing offers many advantages over traditional computing systems; vast parallelism, scalability and robustness among others. A Cellular Automaton (CA) is a biologically inspired computational substrate that has these properties. The most simple CA is an elementary CA, which consists of a one dimensional vector of binary cells.

The cells are only aware of their own value, called a state, as well as the state of neighbor to the left and right. The 1D vector can then be iterated for a number of times, where the next state of a cell is decided by the cells own look-up table, called rule. The rule is a function of the states of the two neighbors and itself. This simple substrate can be utilized for computation.

Reservoir Computing (RC) is a paradigm within Artificial Intelligence (AI) that is used for analysis of temporal data. An RC-system consists of two parts; a dynamical system, called a reservoir, and a classifier, called a readout-layer. Any dynamical system can be utilized as a reservoir, as long as it has some distinct properties that makes it applicable for computation.

CA has these properties, and thus can be used as a reservoir. This combination is called ReCA: Reservoir Computing using Cellular Automata.

In this thesis, the usage of quasi-uniform CA as the reservoir is explored. In a quasi-uniform CA, the 1D vector is split into sub-vectors, where the cells in each sub-vector has the same rule. The sub-vectors are concatenated to make up the full CA vector, such that the last cell in one sub-vector is the neighbor of the first cell in the next sub-vector. This enables information to flow between the sub-vectors. In this thesis, the rules that are applied to each of the sub-vectors are evolved by using an evolutionary process called a Genetic Algorithm.

Many real world problems have sequential data in some way, and RC can be used for processing such data. However, many sequential problems have unknown or varying sequence lengths, which can be a challenge for some RC systems. In this thesis, a framework for applying ReCA systems to sequence-to-sequence learning is proposed for the first time.

The quasi-uniform CA proved very successful on simple problems, where the evolutionary process was allowed to run for a long time. However, when dealing with more difficult problems, the quasi-uniform CA was only able to show a slightly better performance than uniform CA.

The sequence-to-sequence framework worked as intended, but it was not able to demonstrate state-of-the-art performance on well known benchmarks. However, as a proof-of-concept, it did produce promising results, and is a plausible approach for further research on sequence-to-sequence learning.

Sammendrag

Ukonvensjonelle beregningsmodeller kan tilby store fordeler, sammenliknet med tradisjonelle beregningsmodeller. Deriblant enorm parallellitet, skalering og robusthet. En Cellulær Tilstandsmaskin (CA) kan tilby disse fordelene. En elementær CA er en endimensjonal vektor bestående av celler, som hver har en tilstand. Hver celle har oversikt over sin egen tilstand, samt tilstanden til naboen til høyre, og naboen til venstre. Denne endimensjonale vektoren kan itereres, og da blir tilstanden til hver celle oppdatert etter en bestemt regel. Denne regelen er en funksjon av cellens egen tilstand, samt tilstanden til de to nabocellene.

Dynamikkmagasinbasert Beregning (RC) er en teknikk innen Kunstig Intelligens (AI) som kan benyttes til analyse av sekvensiell data. Dynamikkmagasinet må ha visse egenskaper; etterdønningene fra tidligere ytre påvirkninger må gradvis forsvinne fra magasinet, samt at dynamikken i magasinet må være akkurat passe kaotisk. En CA kan benyttes til bruk i RC, og denne kombinasjonen kalles ReCA.

I denne oppgaven benyttes kvasi-uniform CA som magasin. En kvasi-uniform CA har ulike regler for ulike regioner av den endimensjonale vektoren. Regionene grenser til hverandre, og de ytterste cellene er naboer med hverandre. De ulike reglene som benyttes vil i denne oppgaven bli bestemt ved hjelp av en evolusjonær prosess, kalt en Genetisk Algoritme.

Mange av de utfordringene vi omgir oss med i hverdagen har sekvensiell data tilknyttet seg, og RC kan brukes til analyse av nettopp disse. Dersom den lengden på den sekvensielle dataen ikke er kjent på forhånd, kan det skape problemer for enkelte RC systemer. I denne oppgaven blir en rammeverk for å benytte ReCA på slike problemer foreslått.

Bruken av kvasi-uniform CA på enkle problemer var vellykket. Her kunne evolusjonen utvikles over lengre tid, og den fant gode løsninger. På de mer kompliserte problemene var ikke forskjellen på kvasi-uniform og uniform CA særlig stor. Rammeverket for å takle ukjente sekvenslengder fungerte tilfredsstillende. Det klarte ikke å oppnå bransjeledende resultater på kjente problemer. Det fungerte dog godt konseptuelt, og det virker å være et tiltalende område for videre forskning.

Preface

This thesis has been carried out at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU) in Trondheim. It is the conclusion to a five year study in Computer Science.

I would like to thank my supervisor Prof. Stefano Nichele for good guidance within this new and exciting field of Computer Science, and Prof. Gunnar Tufte for stepping in for a brief time when needed. I would also like to thank Andreas Molund for interesting and informal discussions within this specialized field of study.

Simon Borøy-Johnsen and Simen Selseng have both been great companionship during the process of writing this thesis; through both the best of times, and the worst of times.

I would also like to thank my parents, Ellen and Steinar for their help and support over the last five years. It is highly appreciated.

Magnus Skogstrøm Gundersen
Trondheim, June 2017

Table of Contents

List of Tables

List of Figures

Abbreviations

AI	=	Artificial Intelligence
ANN	=	Artificial Neural Network
CA	=	Cellular Automata
GA	=	Genetic Algorithm
RC	=	Reservoir Computing
ReCA	=	Reservoir Computing Using Cellular Automata
TT	=	Time Transition

Chapter 1

Introduction

1.1 Brief background and motivation

Life on earth has existed for billions of years [9]. During this time, new life forms have evolved, while others have perished. The result is nature as we see it today, with its rich variety of species and life forms. How did all this come to existence? It is intriguing to think that Mankind has pondered upon this very question since the dawning of cognitive thinking. As a modern science, this got its genesis by Darwin, when he published ”The origin of species” in 1859 [8]. This proposed a whole new view on the world, and a new view on how nature as we see it today, with all its finesses, have evolved from its very beginning.

It is clear that nature itself cannot be called simple; with its many interactions, it is too complex. If we look in the thesaurus on the word complex, we get the following: ”consisting of interconnected or interwoven parts.” [1]. This leads us into the study of complex systems, where the simple interactions of low-level components are analyzed. If we investigate nature, we often find what seems to be a bottom-up design. For example, the human body is made up of cells, and the cells are made up of atoms.

A compelling next subject is that of Artificial Life (A-life) and Artificial Intelligence (AI). Can a machine think, act and reason like the intelligent creatures we observe in nature? Undoubtedly a question that raises debate, but some argue that yes, they indeed can [33]. Bio-inspired AI is seeking to take inspiration from nature when designing AI-systems, by considering the bottom-up approach. Recent additions to AI is focusing on biologically inspired systems, perhaps with the Artificial Neural Network (ANN) as the largest exponent.

Real life problems like translation, sentence analysis and video analysis often require AI-systems to process time-series data. When processing these data, the AI-system must remember inputs from previous time-steps in order to make correct predictions at the current time-step. A machine learning tool called Recurrent Neural Networks (RNN) can be used to solve this problem [16, Chapter 10].

Unfortunately, training an RNN using traditional methods (i.e. gradient descent) is difficult [2]. The approach called Reservoir Computing (RC) has been proposed [20, 36] to combat this problem. This is done by splitting the RNN into two parts; the recurrent part, called the reservoir, and the trainable feed-forward part, called the readout layer.

In this thesis, an RC-system is investigated, and a biologically inspired computational model called Cellular Automata (CA) [48] is used as the reservoir. This approach to RC, called ReCA, was first introduced by Yilmaz [51], and later verified as a valid approach multiple times [52, 5, 30, 37, 38, 24, 31].

In this thesis, a fully functional ReCA system is implemented in the Python programming language. In the authors specialization project (available as preprint on ArXiv [37], and in press on the journal Complex Systems [1]) it was proposed to use quasi-uniform CA as the reservoir.

1.2 Research goal and research questions

In this thesis, the novel idea of using quasi-uniform CA for ReCA is further researched. In addition to that, the usage of ReCA systems on sequence-to-sequence learning tasks is also researched. The following research goal (RG) is formulated for the thesis:

RG: Doing novel research in the field of quasi-uniform ReCA systems, and the usage of ReCA in sequence-to-sequence learning.

In order to reach this goal, the following research questions (RQ) are formulated:

RQ1: Can quasi-uniform CA be a valuable addition to ReCA systems?

In this thesis, the use of a simple evolutionary algorithm to find the best combinations of rules is proposed. The term quasi-uniform CA describes a CA that has different rules for different group of cells [41]. That means that the reservoir is split into a small, finite, number of regions. Only one rule is applied to the cells within this region, hence the quasi-uniform description.

RQ2: Can ReCA be used for sequence-to-sequence learning tasks?

Many applications of RC involves real-world tasks what has a temporal input and output sequence of unknown length. A good example is translation between languages [44], where the length of both the input-sentence and the output-sentence are unknown before translation. In this thesis, the usage of ReCA on these types of problems is proposed.

[1]`http://www.complex-systems.com/`

1.3 Thesis structure

This thesis is structured in the following way:

- Chapter 1 Introduction: Contains motivation for the research, and gives an introduction to the content of the thesis.
- Chapter 2 Background: Overview of relevant topics and theory in the literature, with focus on why Cellular Automata is a good match for Reservoir Computing systems.
- Chapter 3 Methodology: Detailed description of the implemented system.
- Chapter 4 Experiments and results: Explanation of the experiments, and why they are relevant for the research. Results are presented and discussed.
- Chapter 5 Analysis: Analysis of the ReCA-system, with focus on the stated research questions.
- Chapter 6 Conclusion: Summary of the work related to the research goal, and a conclusion is drawn. Outlines future work.

Chapter 2

Background

This chapter contains the background knowledge that is needed to understand the content of this thesis. First, the study of complex systems and bio-inspired systems is explained, with focus on what advantages they offer. Second, Cellular Automata is covered, with emphasis on why it offers exactly these advantages. Third, Reservoir Computing is explained. Fourth, the usage of Cellular Automata in Reservoir Computing is discussed, with the ReCA system. Finally, the process of an evolutionary algorithm is described, and the usages of evolution together with Cellular Automata.

2.1 Complex and biologically inspired systems

Classical engineering have accomplished great things. Sending astronauts to the moon, submarines to the deepest of oceans and particle collisions at near the speed of light. What does these human engineered systems have in common? They consist of unique and carefully designed components, which are expected to perform flawlessly on a specific task. The unique parts that make up the system, are all necessary in order to make sure the system or product works the way it should.

On the contrary, what seems to be the building blocks of natural systems? They are often made up of a vast number of identical parts or agents, that all work together. In ant colonies, each ant is a very simple creature, but together, they are able to accomplish great things. They behave according to their built-in instincts, and there exist no central organizer that is distributing tasks to the workers; the colony is merely self-organizing. What are the advantages of these type of natural systems?

If one ant were to be missing, it would only have a negligible effect on the colony as a whole. The colony is not dependant on a central controller doing some work before distributing tasks, and can thereby work in a distributed matter. This makes the colony extremely robust. But what happens if the colony were to grow a lot in size? Again, since the ants are self-organized, this would not have an effect on the colony as a whole. The colony is scalable, and can easily handle changes in size.

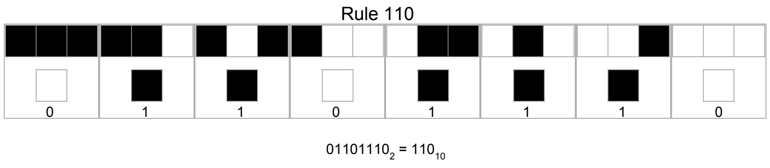


Figure 2.1: Elementary CA rule 110. The figure depicts all the possible combinations that the neighbors of a cell can have. A cell is its own neighbor by convention.

What can we learn from the way nature solves problems? In [10] the authors argue for a new field of research called Morphogenetic Engineering. The goal of this field of research is to exploit the advantages of self-organization and combining these with the human-made, architectured systems.

How can we exploit the advantages of bio-inspired systems? In order to do computation, we need to look at a substrate that has the fundamentals of computation: the transmission, storage and modification of information [27].

Perhaps the largest exponent of bio-inspired artificial systems within computer science is artificial neural networks (ANNs). ANNs are inspired by how the human brain works, and consists of a large number of neurons, that can be connected in various ways. ANNs have proven to be an extremely powerful tool, and have solved problems that traditional machine learning techniques cannot.

Another substrate that fulfills all these requirements is Cellular Automata.

2.2 Cellular Automata

Cellular automaton (CA) is a computational model, first proposed by Ulam and von Neumann in the 1940s [48]. It can be understood as a complex, decentralized and highly parallel system, under which computations may emerge [42]. Some CA have been proved to be Turing complete [7], i.e. having all properties required for computation; transmission, storage and modification of information [27].

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

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

The update-function f, hereafter denoted rule Z, works accordingly with taking three binary inputs, and outputting one binary value. This results in $2^8 = 256$ unique rules, in which the CA may be evolved based upon. An example of such a rule is shown in

Table 2.1: Examples of CA-rules in each of the four Wolfram classes. Each of the four rules is first visualized with a random initial input, and then a single black cell in the middle. The width of the CA-vector is 32, and the rule is applied for a total of 32 iterations.

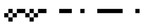
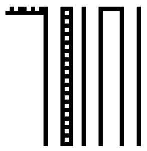
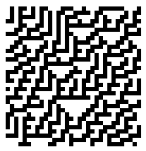
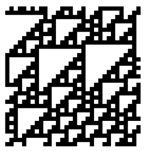
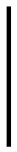
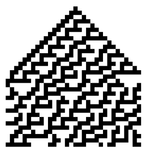
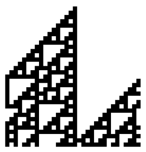
Wolfram cl. I	Wolfram cl. II	Wolfram cl. III	Wolfram cl. IV
Rule 32	Rule 108	Rule 30	Rule 110
			
			

Figure 2.1, where rule 110 is shown. The naming of the rules follow the naming convention described by Wolfram [50], where the resulting binary string is converted to a base 10 number, of which the rule is given its name. The CA is usually updated in lock-steps, where all the cells in the whole 1D-vector is updated at the same time. One update is called an iteration, and the total number of iterations is denoted by I.

The rules may be divided into four qualitative classes that exhibit different properties when evolved [50]; class I: evolves to a static state, class II: evolves to a periodic structure, class III: evolves to chaotic patterns and class IV: evolves to complex patterns. The classes are visualized in Table 2.1. Class I and II rules will enter a repeating pattern after a short while [27], and behave orderly. Class III rules are chaotic, which means that the organization quickly descends into randomness. Class IV rules are the most interesting ones, as they reside at a phase transition between the chaotic and ordered phase, also called *the edge of chaos* [27].

Another approach to CA classification is through the lambda-parameter [27]. This parameter describes the complexity of the rules in a quantitative way, by giving a score between 0 and 1. A value of the lambda parameter close to 0 yields a low complexity of the rule, corresponding to Wolfram class I and, to a lesser degree, class II. A lambda-value of 1 yields a very chaotic CA-rule, and corresponds to Wolfram class III. Wolfram class IV rules have lambda values around 0.5. The rules in this class reside at the phase transition between ordered and chaotic; the edge of chaos.

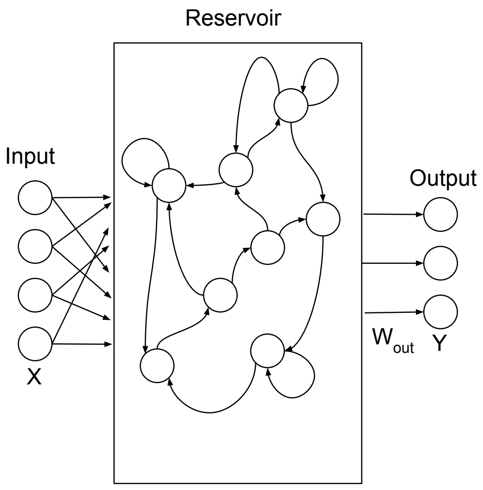


Figure 2.2: General RC framework. Input X is connected to some or all of the reservoir nodes. Output Y is usually fully connected to the reservoir nodes. Only the output-weights W_{out} are trained.

2.3 Reservoir Computing

Feed-forward neural networks are neural network models without feedback-connections, i.e. they are not aware of their own outputs [16, Chapter 6]. They have gained popularity because of their ability to be trained to solve classification tasks that previously were infeasible. Examples include image classification [45], and they were a key part of the system that beat the human champion in the board game GO [40]. However, when trying to solve problems that include sequential data, such as sentence-analysis, they often fall short [16, Chapter 10]. The sentences may have different lengths, and the important parts may be spatially separated even for sentences with equal semantics. Recurrent Neural Networks (RNNs) tackles this problem [16, Chapter 10]. They specialize in processing sequential data, where previous inputs are remembered by the network. This is done by relieving the neural network of the constraint of not having feedback-connections. However, by relieving this constraint, they become notoriously difficult to train by using traditional methods [2].

Most natural systems come in the form of a temporal system, also called a sequential system. This means that an input to the system is dependant on previous inputs. Classical feed-forward architectures have trouble with dealing with these type of tasks [16, Chapter 10].

Reservoir Computing (RC) is a paradigm in machine learning that combines the powerful dynamics of an RNN with the trainability of a feed-forward neural networks. The first part of a traditional RC-system consists of an RNN which is called a reservoir. This reservoir is connected to the second part of the RC-system, which is a feed-forward neural network, which is called a readout-layer. This readout-layer produces the output, and the whole setup can be seen in Figure 2.2.

The field of RC started out as Echo State Networks (ESN) [20] and Liquid State Machines (LSM) [36]. By examining these approaches, important properties of reservoirs are outlined.

Perhaps the most important of these is the Echo State Property [20]. Previous inputs echo through the reservoir for a given number of time steps after the input has occurred, and thereby slowly disappearing without being amplified. This property is achieved in traditional RC-approaches by clever reservoir design. In the case of ESN this means that the connection-weights of the recurrent nodes in the reservoir are scaled appropriately [29].

As discussed in [3], the reservoir must also exhibit edge of chaos [27] behavior. This is in order to guarantee high computational power in the reservoir.

2.3.1 Various RC-approaches

Multiple RC-approaches use reservoirs that have been found to exhibit the properties described. In [12] an actual bucket of water is used as a reservoir, and applied to speech-recognition. In [23], the E.coli-bacteria is used as a reservoir. More recently, the usage of photonic hardware is explored in RC [13]

In [43] and more recently in [4], the usage of Random Boolean Networks (RBN) as a reservoir is explored. RBNs can be considered as an abstraction of CA [14], and is thereby a related approach to the one presented in this thesis.

Evolution-in-Materio is the process of using artificial evolution to directly exploit the properties of physical materials [32]. By utilizing this technique, emergent behaviour in these physical materials can be used for computation, and possibly also as a reservoir in RC. In [11] CA are evolved-in-materio in carbon nanotubes, and is thereby a work that possibly bridges the work in this thesis with Evolution-in-Materio.

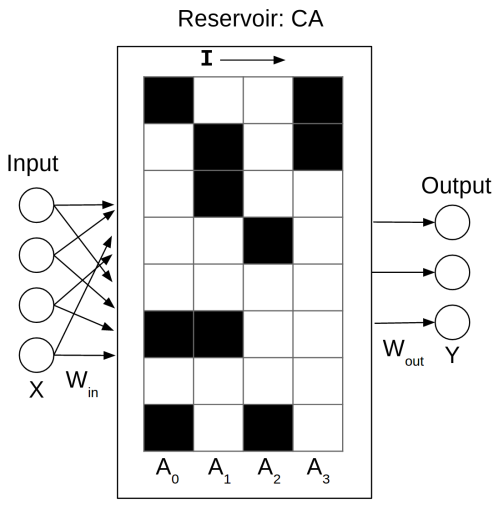


Figure 2.3: General ReCA framework. Input X is projected onto the cells of a one dimensional (1D) cellular automata, and the CA-rule is applied for I iterations. In the figure, each iteration is stored, and denoted by A_i. The readout-layer weights W_{out} are trained according to the target-function. Figure adapted from [52].

2.4 Cellular Automata in Reservoir Computing

As first demonstrated by Yilmaz [51], CA may be used as a reservoir in RC. The conceptual idea of CA in RC is shown in Figure 2.3, and has been dubbed ReCA [30]. The conceptual procedure can be seen i Figure 2.4.

The raw data must be projected onto the CA. This is done by the encoder. If the raw data is non-binary, the encoder must first binarize the data. After the input has been binarized, the CA-reservoir applies the CA rule for I number of iterations.

The iterations of the reservoir can be viewed as the following equations:

$$A_1 = Z(A_0)$$
$$A_2 = Z(A_1)$$
$$...$$
$$A_I = Z(A_{I-1})$$

Where A_m is the state of the 1D CA at iteration m and Z is the CA-rule that is applied. A_0 is the initial state of the CA, often an external input, as discussed later.

As discussed in Section 2.3, a reservoir used for RC should operate at the edge of chaos. Selecting CA-based reservoirs that exhibit this property is trivial. Rules that lie inside Wolfram class IV, as discussed in Section 2.2, will have this property. Additionally, to fully exploit this property, all I iterations of the CA evolution should be used for classification, and can be stated as follows:

$$A = [A_1; A_2; ...A_I]$$

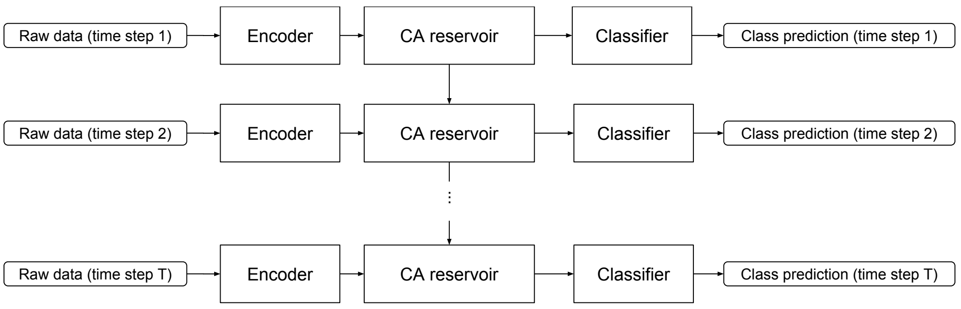


Figure 2.4: General ReCA procedure. Raw data from each time-step is encoded and propagated in the reservoir. The classifier then predicts a class based on the reservoir state. For all time-steps, other than the first, the reservoir state from the previous time-step is also contributing to the reservoir state of the current time-step.

Where A is used for classification.

The ReCA system must also exhibit the echo state property, as described in Section 2.3. This is done by allowing the CA to take external input, while still remembering their current state. As discussed later, ReCA-systems solve this by using some Time-Transition (TT) function, named F, which allows previous inputs to echo through the CA.

2.4.1 Why use CA in RC

As discussed in Section 2.1, the usage of a substrate that allows for vast parallelism, robustness and scalablity is attractive to use for computation. CA offers exactly that. In addition, CA are well understood mathematically, and have a long history of scientific research.

Recent advances in AI have demonstrated considerable increase in performance of AI-systems. State-of-the-art image classification systems can perform at an impressive level [16]. However, the size and computational complexity of such AI-systems can be a challenge. Training the Inception v3 network [46] from scratch takes days or even weeks. Just the weights of a trained version of this network can amount to about 500MB of floating-point numbers. This can be a challenge for some platforms.

Modern type of computing equipment include wearable technology like smart-watches, Google glass and other small devices. In the future, these products are expected to have an increasing degree of AI. These units are often very small in size, with strict energy budgets and low computational power. This is difficult to combine with state-of-the-art AI performance.

Binary Neural Networks (BNNs) have been proposed as a method for reducing the size of an ANN to a size that can be more easy to handle [18]. The BNN is reported to be substantially more power efficient, while at the same time performing at a nearly state-of-the-art level.

BNNs have many similarities with CA, and CA can be seen as an analogy of a BNN. They both rely solely on binary numbers, and can be used for computation. A CA can be viewed as a BNN that only is connected to local nodes, and the rules can be seen as their activation function.

Within RC, CA has been reported to offer additional advantages. Yilmaz [52] reported a speedup of 1.5-3X in the number of operations compared to the ESN [21] approach. Again, this is due to a CA relying on binary operations, while ESN uses floating point operations.

The research on both BNNs and CA suggest that the usage of these techniques can offer comparable performance, with a significantly reduced computational cost. If specialized hardware (e.g. FPGA [17]) or physical substrates (e.g. carbon nanotubes [11]) is utilized, the performance gap can be increased even further.

2.4.2 The details of the ReCA system

Earlier publications show multiple examples of implemented ReCA-systems. Yilmaz [51, 52] created and analyzed a functioning ReCA-system. This system used both elementary CA and Game of Life CA [6] to produce state-of-the-art results. Bye [5] also demonstrated a functioning ReCA-system in his Master's Thesis. Kleyko et al. [24] used a ReCA-system for bio-medical imaging classification. Nichele and Molund [38] successfully demonstrated deep ReCA. McDonald [31] used multiple CA rules for Extreme Machine Learning and Reservoir Computing. In order to understand the parts of a ReCA system, this section describes the details.

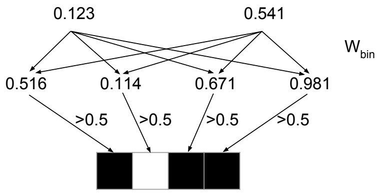
Encoding and mapping In order to apply CA to the data, the data must be projected onto the CA. Since the CA is only binary, the data must first be binarized. This can be done in two ways, as seen in Figure 2.5. By using weighted summation threshold, as seen in Figure 2.5a, the input-number is mapped to all the cells in the 1D CA-vector, with a weight. The resulting value of each cell has a threshold of which decides if its a 1 or 0 [51]. Another option, as seen in Figure 2.5b, is to use a threshold directly on each input-value [24], possibly mapping it to more than one cell. By carefully selecting the threshold values, the floats can be evenly distributed.

After the data has been binarized, the binary values must be mapped onto the CA. It has been experimentally observed that the input should be mapped multiple times [52]. The number of mappings is denoted by the letter R.

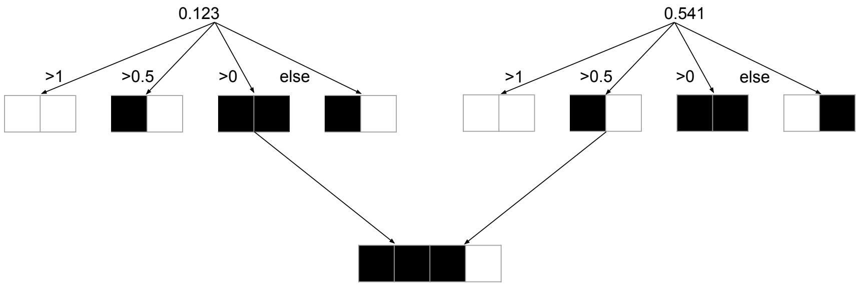
In order to increase the probability of a good mapping, Yilmaz [52] proposes to let the mappings be permutations over the original input. This type of mapping can be seen in Figure 2.6.

It is possible to insert empty cells in the mappings, and thereby padding the input. This will later allow new inputs to be mapped to a CA without destroying previous inputs. This type of mapping can be seen in Figure 2.7. The size of the vector that the input is mapped to is given by the parameter C, that gives the multiple of cells by the input.

Feed-forward or recurrent Yilmaz [52] proposed both a feed-forward and a recurrent design. The difference was whether the whole input-sequence is presented to the system in one chunk or step-by-step. Only the recurrent architectures will be investigated in this thesis. This is because it is more in line with traditional RNNs and RC-systems, and is conceptually more understandable.



(a) Weighted summation threshold. Each input float is multiplied with the W_{bin} matrix, and summed, giving a value for each of the cells in the CA. The value is then thresholded to binarize. In this example, the value of the cell is 1 if the weighted sum is above 0.5, and 0 if it is not.



(b) Direct threshold quantizing. The line with the highest valued inequality is chosen. The values of the inequalities can be adjusted to evenly distribute the floating point numbers in the training-set, and thereby quantizing the data. Note the Grey-encoding of the CA-vector.

Figure 2.5: Binarization schemes.

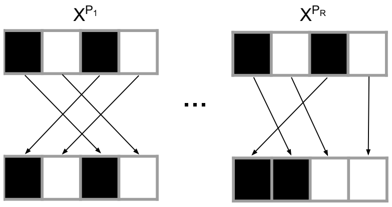


Figure 2.6: Random permutation mapping. For a total of R permutations, X is randomly mapped to vectors of the same size as the input-vector itself.

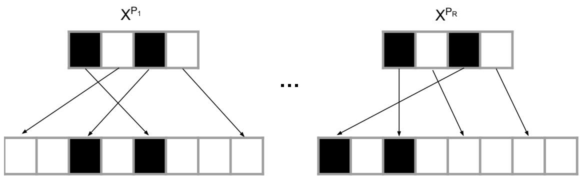


Figure 2.7: Random permutation with C-padding. The input X is randomly mapped to a vector with size larger than the input vector itself. This mapping is done R times. The size of the vector that the input is given by the C-parameter, which the size is given by $C * |X^{p_n}|$. In this case $C = 2$.

Concatenation of the encoded inputs before propagating into the reservoir or not After the random mappings have been created, these must be distributed to the CA. In the recurrent architecture, Yilmaz [52] concatenate the R number of permutations into one large vector of length ($R * input\_length$) before propagating it in a reservoir of the same width as this vector. The 1D input-vector at time-step t can be expressed as follows:

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

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

$$A^t = [A_1; A_2; ...A_I]$$

Bye adapted a different approach, the same one that was also used by the feed-forward architecture of Yilmaz. Here, the R different permutations are iterated in separate reservoirs, and the different reservoirs are then concatenated before they are used by the classifier. The vector which is used for classification at time-step t is as follows:

$$A^t = [A_{P_1}^t; A_{P_2}^t; ...A_{P_R}^t]$$

Where $A_{P_n}^t$ is the vector from the concatenated reservoir.

Time-Transition function In order to allow the system to remember previous inputs, a Time-Transition (TT) function is needed to make the transition between the previous time-step and the current.

If the encoding was done without any empty cells inserted into the CA (i.e. $C = 0$), the TT function must be a bit-wise function, as shown in Figure 2.8. Yilmaz [52] first proposed normalized addition as TT-function. This function works in the following way: The cell values are added, and if the sum is 2 (1+1) the output-value becomes 1, if the sum is 0, the output-value becomes 0 and if the sum is 1, the cell-value is decided randomly (0 or 1). The initial 1D-CA-vector of the reservoir at time-step t is then expressed as:

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

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

If the encoder has inserted any empty cells into the mappings (i.e. $C > 1$), another TT-function can be used.

The TT-function dubbed Mapping Copy inserts data bit-wise from new time-steps at the same cells as the original mapping done by the encoder. The cells that do not have any mappings to them remain in the same state as in the last iteration of the previous time-step. This procedure can be seen in Figure 2.9. Bye [5] introduced the method.

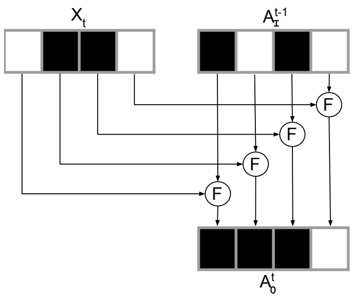


Figure 2.8: Bit-wise Time-Transition function. The sequence input X_t is combined with the state of the reservoir at the last iteration of the previous time-step A_I^{t-1}. The function F may be any bit-wise function. Only one permutation is shown in the figure to increase readability.

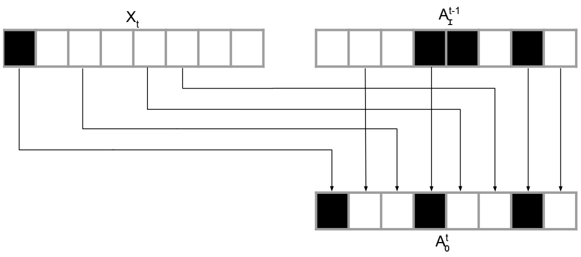


Figure 2.9: Time-Transition function Mapping Copy. The input is directly copied from X_t, according to the mapping from the encoder, as shown in Figure 2.7. The other cells have their values copied from the last iteration of the previous time-step A_I^{t-1}. Only one permutation is shown to increase readability.

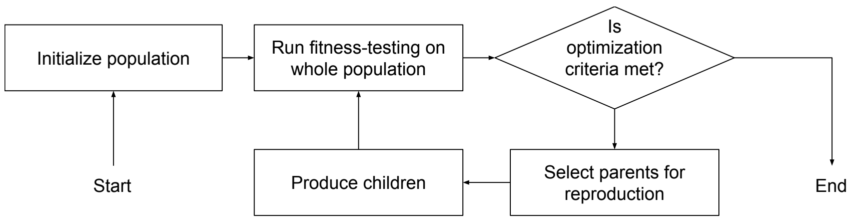

Initialize population
Run fitness-testing on whole population
Is optimization criteria met?
Start
Produce children
Select parents for reproduction
End

Figure 2.10: Genetic Algorithm procedure.

2.4.3 Complexity measure of ReCA systems

To be able to compare different ReCA-approaches, we need a concept of how complex the CA-reservoirs are. An obvious metric is to use the size of the CA-reservoir itself, that is, the number of cells used in each simulation of the reservoir. This will give a useful metric when comparing the different approaches, and also to the system that was developed in this thesis.

In [15] quality metrics of traditional RC-system are unified. Separation ratio is introduced as a method for reviewing the quality of the reservoir. This gives a good indication of whether the reservoir will have the required property; not being too chaotic, not too static, but residing at the edge of chaos. When using CA as the reservoir, it is not clear how to apply the same techniques. However, in CA we have a concept of Wolfram classes which describes the behaviour of the CA, as discussed in Section 2.2. In addition to this we have the lambda-parameter [27] which precisely describes the CA behaviour. However, the lambda parameter might not be properly defined for all CA [27].

2.5 Evolutionary algorithms

Evolutionary Algorithms (EAs) are a group of bio-inspired algorithms that are designed to perform advanced probabilistic searches [49]. The strength of this approach is that minimal domain knowledge is needed, and it is a good fit for investigating problems that have solutions with unknown characteristics.

2.5.1 Genetic Algorithms

The perhaps best known evolutionary algorithm uses genetic operators such as genotype/phenotype and population in the system, and is referred to as Genetic Algorithms (GAs). The general procedure in a GA can be seen in Figure 2.10. In order to successfully apply GAs, an understanding of the key components of such a system is needed.

Adult selection

The adult selection scheme is concerned with how adults are selected for the next generation. One option is to have full replacement, i.e. removing all individuals from the previous generation when starting on the next. Another option is to have full mixing. This means that the best individuals from the pool of both adults and children are selected.

Elitism

To ensure that well performing individuals are not discarded, a technique called elitism can be implemented. This means that a number of the best individuals from one generation is directly copied to the next.

Parent selection

Parent selection concerns which individuals are selected for reproduction. One option is the Roulette style parent selection. This can be explained that each individual gets the sector of a roulette corresponding to its fitness. The roulette wheel is then spun, and the arrow stops at a random individual. The individuals with higher fitness will then have a larger probability of being selected because it has a larger sector that the arrow can stop. Another option is to use the Tournament style parent selection. A number of individuals are randomly picked from the population, and ranked according to their fitness-value. The best individual in the tournament is then selected with a high probability, the second best with a lower probability, and so on.

Reproduction: crossover and mutation

Once two individuals are selected for reproduction, their genotypes has to be joined. This can be done in multiple ways. Single point crossover will split both the genotypes at the same location, and create two new genotype that consist of a portion of both genotypes. This portion is called the crossover rate. If the crossover rate is 0, the new genotypes will be copies of the parent genotypes. After the new genotype has been made, mutation can happen. With a low probability, called mutation rate, the new genotype can be altered. This is done to allow new combinations of genotypes to emerge, and to make the search more stochastic.

2.5.2 Evolution in CA

Evolutionary processes can be successfully applied to optimization of ANNs [35]. This yields a turn towards more evolutionary based optimization processes in AI, and gives a good motivation for using evolution with CA. The usage of a GA to evolve CA rules is not new. Mitchell et al. [34] used GAs for density classification of a bit string, where the system is asked to determine if there is a majority of 1's or 0's. This problem might seem trivial, but the problem is difficult for system like a CA that does not have a central controller.

Chapter 3

Methodology

In order to explore the usage of CA in an RC-system, a functioning ReCA-system has to be implemented. The developed system is a continuation of the system in [37], and the code base is available online [1]. It gives a foundation for exploring the main research goal of this thesis; doing novel research in the field of quasi-uniform ReCA, and explore the usage of ReCA in sequence-to-sequence learning.

The implemented system consist of a CA-simulator and an RC-framework. This section contains a description of both of these parts, and a description on how they work together to form the ReCA-system. Additionally, the evolutionary process for evolving rules is explained.

3.1 CA-simulator

In order to use CA as a substrate for use in RC, a functioning CA-simulator had to be implemented. The simulator is able to take a binary vector of an arbitrary length, and iterate the vector for a number of iterations. The rule that is applied to the vector can be identical over the whole string, one rule per cell, or anything in between. This can be seen in Figure 3.1, where both a uniform rule is shown, and a quasi-uniform rule-set, with four rules.

The CA is visually inspected to be correct. This visualization gives valuable insights into how the CA behaves when parameters are changed, and makes it easier to detect bugs or quirks with the implementation.

[1] `https://github.com/magnusgundersen/master-thesis`

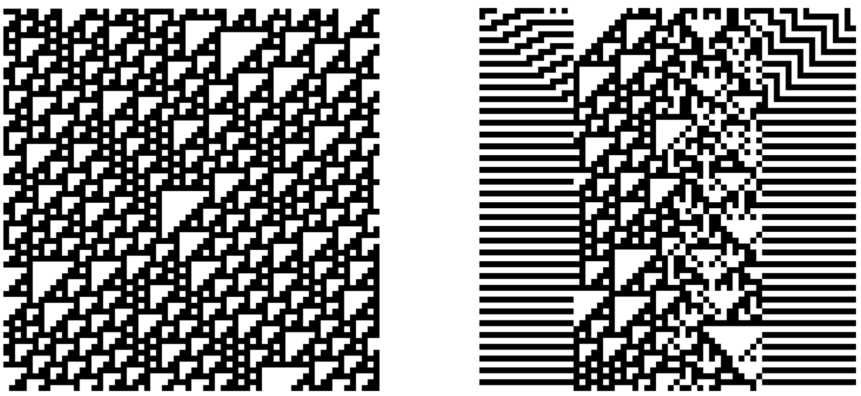

(a) Elementary rule 110 **(b)** Four elementary rules; 11, 110, 90 and 85.

Figure 3.1: CA simulator visualizations of a 1D vector of size 64 that is iterated 64 times.

3.2 RC-framework

The RC-system was developed from scratch. The system is an independent framework that can be used for any reservoir. The framework consists of a fit procedure and a predict procedure. During the fit procedure, all the training-set examples are propagated in the reservoir, and the results of this propagation is mapped to the correct prediction. The classifier is then trained on the resulting set. During the predict procedure, the input is propagated in the reservoir and directly fed to the trained classifier. The classifier output is then produced and the next step is decided accordingly.

The RC-framework has a range of available classifiers from the Scikit-learn library[39], including the Linear-SVM and PerceptronSGD.

3.2.1 Sequence-to-sequence learning

If the problem is a fixed-length sequence problem or a classification task, it is already known how long the sequence will be. If the problem is a sequence-to-sequence problem, an addition to the framework is needed. In order to allow the system to take an arbitrary length on both the input and the output sequence, the RC-framework must be adapted for this. This is done by allowing the classifier to give signals both for prediction start, and prediction end, in addition to the output-signals that is required for the task. An example of such a data set can be seen in Figure 3.2. Only the inputs x_i and outputs y_j is relevant for the task. The end of the input-sequence is signaled by the eos-signal, and the p-signal indicates that the prediction shall start. The p-signal is then a static input-signal that can be repeated as many times as needed.

Before the prediction signal is given, the RC-framework is indicating that it is waiting by turning on the w-signal. After the output-data is transmitted with the y_j-outputs, the prediction end signal pes are turned on by the system, and the sequence is finished. To prevent the system from going into an infinite loop of wrong predictions, an upper limit of prediction time-steps must be set.

x_1	x_2	eos	p	y_1	y_2	w	pes
1	1	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	1	0	0	0	0	1	0
1	0	0	0	0	0	1	0
0	0	1	0	0	0	1	0
0	0	0	1	1	0	0	0
0	0	0	1	0	1	0	0
	. . .			0	1	0	0
0	0	0	1	0	0	0	1

Figure 3.2: The figure shows two input data-signals and two output data-signals. The end of sequence-signal eos signals that the input is finished, and the predict-signal p signals that the prediction can start. The output has a wait signal w, which indicates that the output is waiting for the input-sequence to finish. After the eos signal has been fired, the prediction in the data-channels start. The system signals that the prediction is finished by turning on the prediction end signal pes.

3.3 ReCA system

The ReCA-system consists of the CA-simulator inserted as the reservoir of the RC-framework. The full architecture can be seen in Figure 3.3.

The tasks where the raw data is represented as floats, the data needs binarization. The binarization is done by direct threshold quantizing, as described in Section 2.4.2.

The encoding that uses the C-parameter, as described in Section 2.4.2, is used. However, there is one difference. The mappings are not permuted, i.e. they are not random. This results in a deterministic mapping, where the input-vector, which size is denoted by N, is repeated R number of times. Empty cells according to the value of C is then injected between each of the cells. The reason for this is to reduce the randomness of the mapping, and thereby making the ReCA system more stable. Also, since the usage of quasi-uniform CA allows for different treatment of each mapping, the random permutations are considered unnecessary.

After the input is encoded, the vectors are concatenated, as seen in Figure 3.3. The resulting vector is iterated for I iterations. The TT function was chosen to be Mapping Copy, as described in Section 2.4.2. Since the encoding is without permutations, the copy from each time-step to the next is straightforward. This procedure is repeated for all time-steps.

The resulting ReCA space-time diagram can be seen in Figure 3.4. This type of diagram will be used throughout this thesis to visualize the ReCA-systems behaviour. The visualizations are created with the Python plotting library Matplotlib [19].

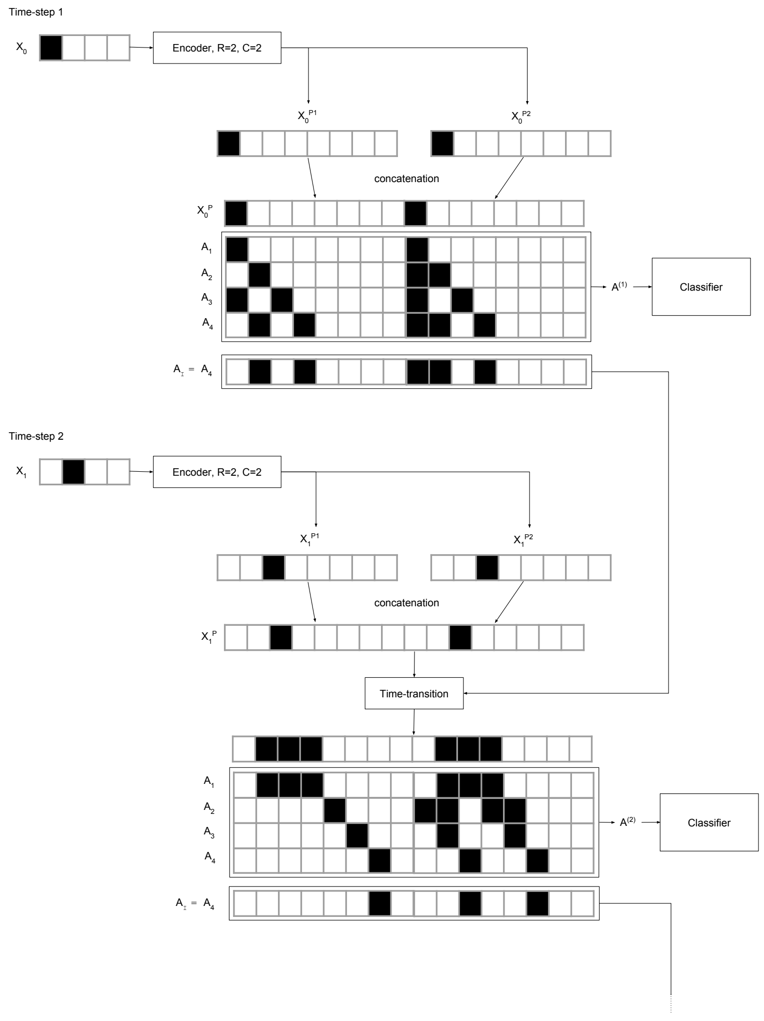
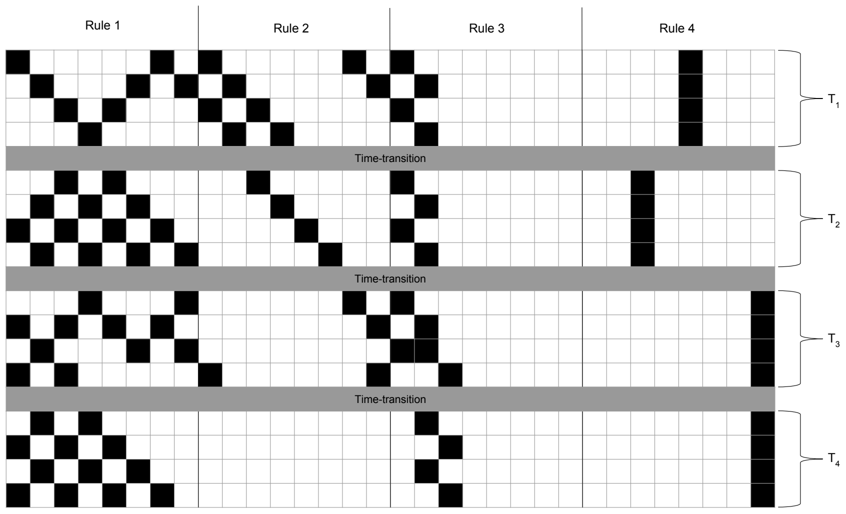
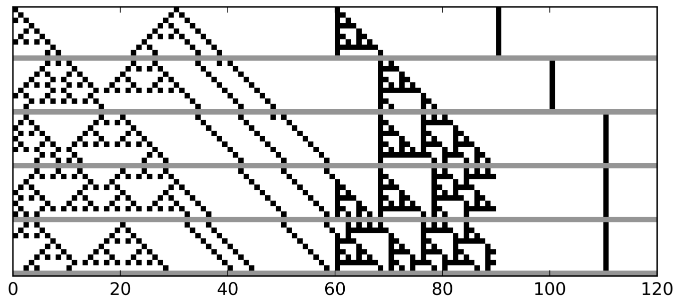


Figure 3.3: Schematic of the flow in the implemented system. The input-vector is of size $N = 4$. The encoding is exemplified with $R = 2$ and $C = 2$, which yields a size of eight for each permutation. The two permutations are then concatenated. At time-step 1, there are no previous inputs, and the concatenated vector is simply used as the first iteration of the CA-reservoir. The rule Z is then applied for I iterations. At time-step 2, the encoding and concatenation is repeated. The Time-Transition scheme is then applied. The procedure as described in time-step 2 is repeated until the end of the sequence.



(a) Conceptual schematic of the ReCA space-time diagram. Four different rules are applied to eight cells each, making the total CA width of 32. Rule 1 is a Wolfram class III/IV type of rule, and is creating complex dynamics. Rule 2 is a side-shifting rule that is carrying information across to rule 3. Rule three is at T_3 colliding with the information from rule 2. Rule 4 is merely copying cells, without an obvious meaning.



(b) Actual ReCA space-time diagram from the implemented system, perturbed by a dummy-signal. The rule-set is the following: $[90, 16, 60, 110]$. ReCA-parameters: $N = 3$, $R = 4$, $C = 10$, $I = 8$. The resulting width of the CA-vector is 120.

Figure 3.4: Visualizations of the ReCA space-time diagrams. The 1D CA-vector is divided into four sub-vector, each with its own rule. The gray area between the CA-iterations is the Mapping Copy Time-Transition function, as described in Section 2.4.2.

Table 3.1: GA base parameters used for the experiments in this thesis.

Crossover type	Two point crossover
Crossover rate	Stochastic $(0 - 1)$
Mutation rate	0.15
Mutation bit flips prob.	[0.67, 0.22, 0.11]
Parent selection	Tournament
Tournament size	5
Tournament selection prob.	[0.70, 0.20, 0.09, 0.01]
Adult selection	Full generational mixing
Elitism	5 best individuals

3.4 Evolving quasi-uniform rule-sets for ReCA systems

3.4.1 Quasi-uniform CA for ReCA

As first proposed in the paper [37], a set of rules, coupled together might enhance the performance of the ReCA-system, especially for difficult tasks. In this thesis the usage of an evolutionary algorithm as a tool for finding these rules is introduced.

By having different rules in the reservoirs, one might be able to solve different aspects of the same problem, or even two problems at the same time. In [5], both the temporal parity and the temporal density task is solved at once by using parallel, non-coupled, reservoirs.

Which rule is most suited for a task still remain somewhat unexplored. The characteristics and classes described in Section 2.2 are useful knowledge, however it does not precisely describe why some rules perform better than others on different tasks. As seen in [5, Table 5.1], different rules proved useful on a variety of task.

3.4.2 GA for evolving quasi-uniform CA rule-sets

The goal of the evolutionary search in this thesis was to show that a simple GA can be used successfully in ReCA-systems. As always in AI and machine learning, choosing the best hyper-parameters can be a true challenge. In order to keep the time used for tuning the GA to a minimum, some experimentation was done in order to decide the values. Table 3.1 shows the hyper-parameters of the GA that was common for all the experiments in this thesis. The genotype of the individuals was the rule-set itself. There is a constraint on how many distinct rules were allowed on each evolutionary run. The rule-set is encoded as a bit string, with each rule represented as eight bits (256 different rules).

When doing reproduction of a new individual, a two point crossover scheme was used. This can be seen in Figure 3.5. The two crossover point were in all experiments chosen at random, with a stochastic crossover rate between 0 and 1. This was done because it allowed one part of a rule set to be injected into another rule set. This allowed for two good solutions to possibly be combined into a new one.

The Tournament selection scheme for parent selection was chosen because it was experimentally observed that it gave good results. As seen in Table 3.1, size of the tournament was chosen to be 5. The tournament probability distribution, i.e. the probability the

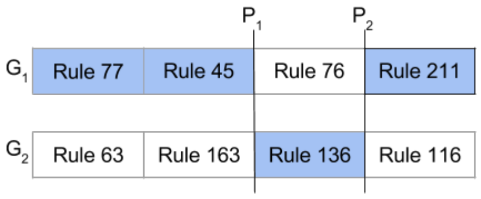


Figure 3.5: Two point crossover. The resulting genotype is the rule set marked in blue.

that best individual was chosen was 0.70, the second best 0.20 and so on. These number were chosen to ensure that the diversity of the rule-sets were not lost.

The mutation rate was set to 0.15, i.e. with the probability of 0.15, the genotype bit-string was mutated. In Table 3.1 the probability of the number of bit flips is given. 0.67 probability of one bit flip, 0.22 probability of two independent bit flips, and a 0.11 probability of three independent bit flips.

For all the experiments in this thesis, the fitness evaluation is the most time-consuming part of the evolutionary run. The procedure for fitness-testing was to test the individual rule-sets on the problem itself. It was experimentally observed that the rule-sets had to be tested many times. The fitness evaluation can be viewed as the following equation:

$$fitness = \frac{\sum^{n}((correct\_predictions/total\_examples) * 1000)}{n}$$

Where n is the number of identical tests on the problem. Each individual had to be tested many times on the same problem because the solutions proved to be unstable. To further combat this instability, the individuals were retested if they achieved a certain fitness value from the n first tests. They were then tested m more times, and the new fitness-score was then given.

3.5 Measuring computational complexity of a CA-reservoir

As stated several times in [52], the size of the reservoir was a crucial part of the success of the system. The size will be measured by how many mappings (R), how many empty cells were injected (C), and how many iterations were done (I).

The size of the reservoirs will remain the same both for the one-rule reservoirs and the quasi-uniform reservoirs. This is crucial in order to be able to directly compare their performance.

The CA reservoirs will also be classified according to their Wolfram classes. For the quasi-uniform reservoirs, this means that the rules that make up the rule-set will be classified.

Chapter 4

Experiments and results

This chapter contains five experiments where a rule-set is evolved for a specific task. Each experiment has a description of the task, a strategy for evolving a rule-set and ten evolutionary runs. The best evolved quasi-uniform rule-set is then tested against all uniform CA rules, and compared.

At the end, a proof-of-concept is given for how the ReCA system can be used for machine translation.

4.1 Evolving quasi-uniform CA rule-sets for synthetic learning tasks

This section contains the experiments that are chosen to investigate the first research question: **RQ1: Can quasi-uniform CA be a valuable addition to ReCA systems?** For each experiment, both the input-sequence length and the output-sequence lengths are known.

4.1.1 5-bit problem

Problem description

This task is designed to test the long-short-term memory of the system, and has become a benchmark for RC-systems. It is included in this thesis to test the capability of the system to perform on a simple short-term memory task.

An example data-set from this task is presented in Figure 4.1. The length of the sequence is given by T. a_1, a_2, a_3 and a_4 are the input-signals, and y_1, y_2 and y_3 are the output-signals. At each time-step t only one input-signal, and one output-signal, can have the value 1. The values of a_1 and a_2 at the first five time-steps give the pattern that the system shall learn. The next T_d time-steps make up the distractor-period, where the system is distracted from the previous inputs. This is done by setting the value of a_3 to 1. After the distractor period, the a_4 signal is fired which marks the cue-signal. The system is then

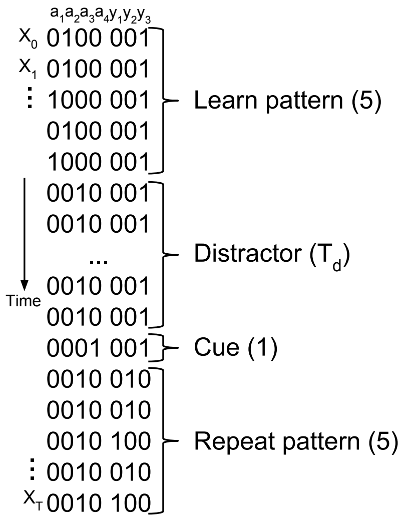


Figure 4.1: Example data from the 5-bit task. The length of the sequence is T. The signals a_1, a_2, a_3 and a_4 are input-signals, while y_1, y_2 and y_3 are output-signals. In the first five time-steps the system learns the pattern. The system is then distracted for T_d time-steps. After the cue-signal is set, the system is expected to reproduce the pattern that was learned.

asked to repeat the input-pattern on the outputs y_1 and y_2. The output y_3 is a waiting signal, which is supposed to be 1 right until the input-pattern is repeated. For details on the task, please see [21].

Evolving a solution

The 5-bit task was evolved with the procedure as described in Section 3.4.2. Table 4.1 shows the ReCA parameters used for the 5-bit task. The values were chosen through trial and error, and were deliberately chosen to be as low as possible, with the GA still being able to find a good performing individual. A total of ten evolutionary runs were done with the parameters in Table 4.2.

All but one evolutionary run reached a fitness value of 1000, as seen in Table 4.3. However, the number of generations that was needed varied significantly, ranging from 16 to 921. After the GA runs were finished, the best individuals were retested 120 times to validate their fitness. Only Run 7 did not have a drop in performance from the GA-fitness testing to the validation testing.

In Figure 4.2 we can see the GA-runs visualized. We can observe that the GA found solutions that were local optima before finally converging towards the global optimum.

Table 4.1: ReCA parameters for the 5-bit task. The usage of the parameters is described in Section 3.3.

N	4
R	4
C	4
I	2
Classifier	Perceptron SGD
Time-Transition function	Mapping Copy
Permute mappings	No
Training examples	32
Testing examples	32
Distractor period	10

Table 4.2: 5-bit specific GA parameters. Base parameters can be seen in Table 3.1

Population size	14
Max no. generations	1000
Fitness threshold	1000
No. fitness tests (n)	4
No. fitness retests (m)	10
No. rules in genotype	4

Table 4.3: Results from the GA runs performed on the 5-bit task with $T_d = 10$.

Run no.	Gens.	Best fitn.	Best rule set	Validation testing fitness
1	26	1000	[28, 170, 85, 179]	988 ± 93
2	234	1000	[195, 114, 187, 226]	888 ± 224
3	111	1000	[51, 125, 26, 90]	985 ± 86
4	16	1000	[31, 98, 195, 60]	988 ± 42
5	1000	913	[105, 106, 154, 170]	743 ± 185
6	16	1000	[160, 81, 32, 102]	985 ± 40
7	289	1000	[105, 90, 216, 13]	1000 ± 0
8	921	1000	[145, 195, 153, 235]	954 ± 156
9	114	1000	[58, 194, 85, 60]	968 ± 89
10	32	1000	[197, 72, 87, 60]	990 ± 44

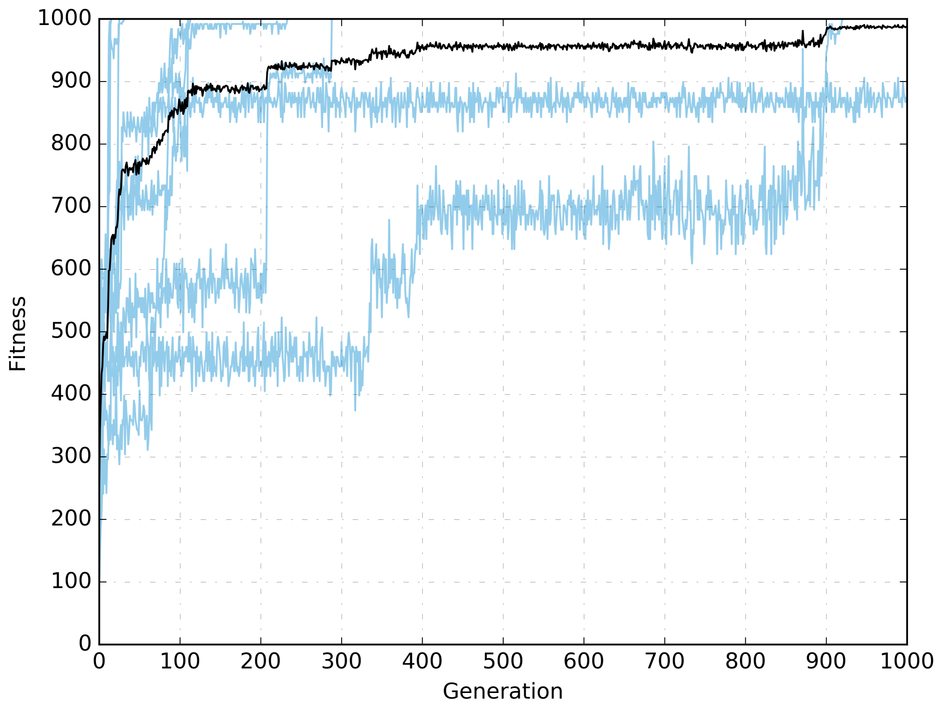
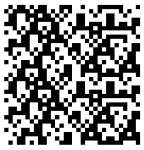
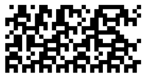
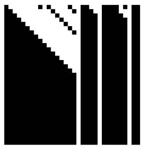
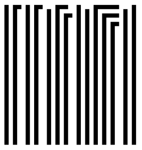
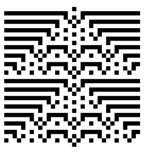
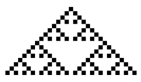
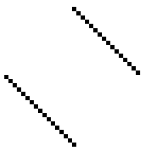
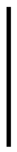


Figure 4.2: Fitness plot from the ten GA-runs to evolve a rule-set for the 5-bit task. The light blue lines represent each of the evolutionary runs, and the black line is a per-generation mean fitness across all ten runs.

Table 4.4: Fitness score of the tested rules on the 5-bit task with $T_d = 10$. Only the five best best performing uniform rules are shown.

Rule	Wolfram Class	Fitness score
Run 7	II & III	1000 ± 0
28	II	59 ± 7
69	II	59 ± 8
70	II	59 ± 9
93	II	59 ± 8
156	II	58 ± 7

Table 4.5: Rules evolved in Run no. 7, 5-bit task with $T_d = 10$. The table shows which Wolfram class the rules belong to, a visualization of the rule with a random initial state, and a visualization of the rule evolved with single black cell in the middle.

Rule 105	Rule 90	Rule 216	Rule 13
Wolfram cl. III	Wolfram cl. III	Wolfram cl. II	Wolfram cl. II
			
			

Results from validation testing

All 256 uniform rules were tested 120 times against the same fitness-measure as was applied in the GA. The ReCA parameters were the same, and can be seen in Table 4.1, with one exception. The uniform rules have the permutation of mappings turned on, while the evolved rule-sets has it turned off. As seen in Table 4.4, the evolved rule-set significantly outperformed the uniform rules.

Discussion

Why did the quasi-uniform solution outperform the uniform rules? By investigating the best performing rule-set from the GA-runs (Run 7), some insights can be drawn. This rule-set is visualized in a space-time diagram in Figure 4.3, and the rules that make up the rule-set are visualized one by one in Table 4.5. The quasi-uniform rule-set consists of rules from both Wolfram class II and III. As seen in Figure 4.3, the two rules on the left side (105 and 90) create complex dynamics. The third rule (216) is on multiple occasions branching a line between the complexity on one side, and the fourth rule (13) on the other. The fourth rule does sometimes reach an attractor-pattern.

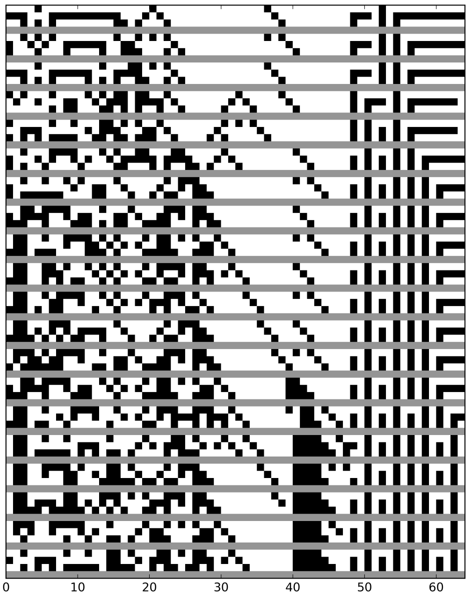


Figure 4.3: Example run of the evolved rule-set from GA Run 7. The input-signal is $[a_2, a_2, a_2, a_1, a_1]$. Size of the reservoir is $(N = 4) * (R = 4) * (C = 4) = 64$. The figure-style is explained in Figure 3.4.

The uniform rules that were found to perform best were surprising. However, the results are poor, with the best rule having a fitness of only 59, such that one might not put too much into the uniform rule results.

On the other hand, in the quasi-uniform rule-set, familiar rules appear. Both rule 90 and rule 105 have been previously proven to be able to solve the 5-bit task [52, 37]. It seems like the evolved rule-set shows a new dynamic between the rules that enables the reservoir size to be massively reduced, without the loss of accuracy.

4.1.2 20-bit problem

Problem description

The 20-bit problem is similar to the 5-bit problem, but more difficult [21]. The data is on the same format as seen in Figure 4.1, but the input-signals now range from a_1 to a_7, and the output-signal from y_1 to y_5. The learn-pattern time-frame is increased from five to ten. This gives a total information capability of 20-bit [21]. Details on the task is available in [21].

Evolving a solution

The 20-bit problem was then applied to the GA, as described in Section 3.4.2. The GA hyper-parameters can be seen in Table 4.7. The maximum number of generations was

Table 4.6: ReCA parameters used for the GA-runs in the 20-bit task. The usage of the parameters is described in Section 3.3.

N	7
R	6
C	16
I	2
Classifier	Perceptron SGD
Time-Transition function	Mapping Copy
Permute mappings	No
Training examples	120
Testing examples	100
Distractor period	10

Table 4.7: 20-bit specific GA parameters for the 20-bit problem. Base parameters can be seen in Table 3.1

Population size	14
Max no. generations	150
Fitness threshold	1000
No. fitness tests (n)	4
No. fitness retests (m)	10
No. rules in genotype	6

decreased to 150. This was done because the 20-bit task is much more computationally demanding, and therefore needed much more computing time. The number of rules were increased to six, in order to follow the number of mappings R.

The ReCA parameters as seen in Table 4.6 were chosen by trail and error. Since the 20-bit problem is regarded as a more difficult problem than the 5-bit problem, the values of R and C were drastically increased. The I was kept at 2 because the impact of number of iterations is not clear, and the fact that deciding to leave it at 2 meant that the hyper-parameter search space got reduced.

As seen in Table 4.8, none of the runs reached a fitness of 1000. Note also that the solutions were a lot more unstable, with a large standard deviation. The solutions also had a noticeable drop in performance from fitness testing to validation testing.

As seen in Figure 4.4, the GA-runs had a more gradually increase in fitness then what was observed in the 5-bit problem. The trend in the mean-graph is positive.

Table 4.8: GA runs performed on the 20-bit task with $T_d = 10$.

Run no.	Gens.	Best fitn.	Best rule set	Validation fitn.
1	150	582	[43, 60, 199, 11, 81, 53]	354 ± 151
2	150	580	[127, 160, 10, 174, 212, 72]	421 ± 109
3	150	552	[244, 199, 56, 26, 10, 46]	398 ± 106
4	150	745	[11, 34, 81, 17, 138, 166]	601 ± 157
5	150	542	[84, 74, 81, 166, 102, 244]	502 ± 90
6	150	662	[11, 43, 66, 174, 102, 112]	525 ± 106
7	150	710	[17, 102, 240, 11, 81, 166]	572 ± 140
8	150	645	[174, 236, 59, 244, 84]	506 ± 102
9	150	675	[174, 166, 84, 180, 116, 170]	590 ± 104
10	150	540	[81, 17, 244, 227, 200, 60]	439 ± 147

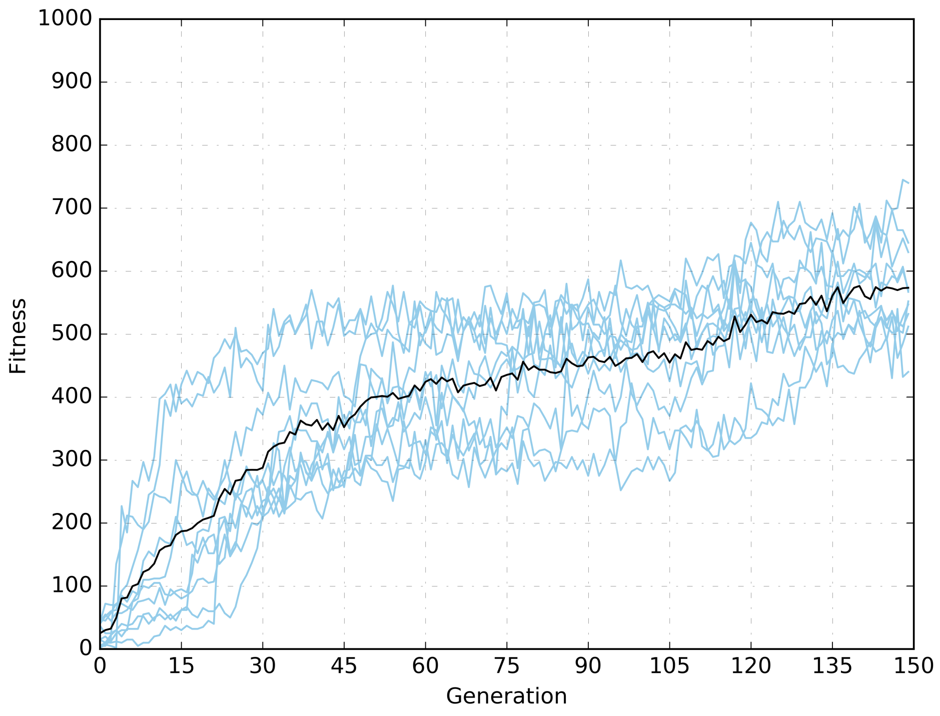
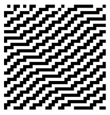
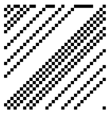
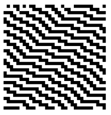
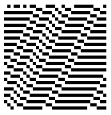
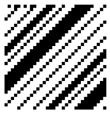
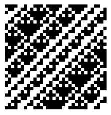
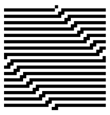
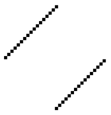
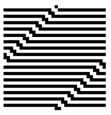
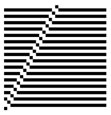
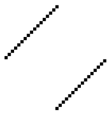
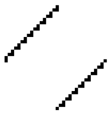


Figure 4.4: Fitness plot for the ten GA runs for the 20-bit problem. The light-blue lines represent one GA run, and the black line is a per-generation mean over all ten runs.

Table 4.9: Fitness score of the tested rules on the 20-bit task, with $T_d = 10$. Only the best five best performing uniform rules are shown.

Rule	Wolfram Class	Fitness
240	II	918 ± 259
170	II	916 ± 252
85	II	893 ± 283
174	II	888 ± 88
244	II	885 ± 99
...	..	...
Run 4	II	601 ± 157

Table 4.10: Rules evolved in Run no. 7, 5-bit task with $T_d = 10$. The table shows which Wolfram class the rules belong to, a visualization of the rule with a random initial state, and a visualization of the rule evolved with single black cell in the middle.

Rule 11	Rule 34	Rule 81	Rule 17	Rule 138	Rule 166
Wolfr. cl. II	Wolfr. cl. II	Wolfr. cl. II	Wolfr. cl. II	Wolfr. cl. II	Wolfr. cl. II
					
					

Results from validation testing

The evolved solution was then tested against the best performing uniform rules.

All 256 uniform rules were tested 120 times against the same fitness-measure as was applied in the GA. The ReCA parameters were the same, and can be seen in Table 4.6. The uniform rules have the permutation of mappings turned on, while the evolved rule-sets has it turned off. As seen in Table 4.9 the best evolved rule-set was outperformed by the uniform rules. Note also that all the best performing uniform rules are all in Wolfram class II.

Discussion

The uniform rules that proved best on this task were surprising, and different from the rules was found best in the 5-bit task. When examining the visualization of the evolved rule in Figure 4.5, we can observe that the evolved solution indeed looks different for the 20-bit problem, then what we observed for the 5-bit problem.

As seen in Table 4.10 all the rules in the evolved rule-set were in Wolfram class II. It seems like they side-shift the input.

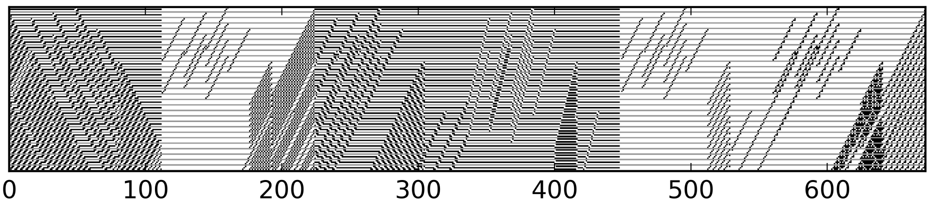


Figure 4.5: Example run of the evolved rule-set from GA Run 4 for the 20-bit problem. The figure-style is explained in Figure 3.4.

4.1.3 Dual problem

Problem description

The Dual problem is new in this thesis, and is designed to test whether a problem that consist of two sub-problems can be more easily be solved by a quasi-uniform CA. The hypothesis is that the different rules might solve different aspects of the same problem.

The 5-bit problem as seen in Section 4.1.1 is combined with the temporal majority problem (a temporal variation of the density classification task) [34]. Example data from the dual problem can be seen in Figure 4.6.

The channels a_1, a_2, a_3 and a_4 and y_1, y_2, y_3 are ordinary 5-bit problem, as described in Section 4.1.1. The density-channels d_1, d_2, d_3 transmits data in the first five time-steps, and are then 0 for the rest of the sequence. Five time-steps after the cue-signal a_4, the system uses the output-channel m_1 to signal that the d-channels had a majority of 0's, and m_2 if they had a majority of 1's.

The data-set is generated by repeating the 32 5-bit signals as many times as needed, and for each of the density-signals set the value randomly to either 0 or 1. Since the number of density signals is odd, the problem is well defined.

Evolving a solution

Both the ReCA parameters and the GA-parameters were chosen to be the same as with the 20-bit task. These can be seen in Table 4.6 and 4.7. The parameters were chosen to be the same as with the 20-bit problem because it was considered to be of somewhat same difficulty scale.

The dual problem proved to be a more difficult task than the 20-bit task. Table 4.11 shows the results from the GA runs. None of the ten runs reached a fitness-value of 1000. All runs show a significant drop in fitness from fitness-testing to validation-testing.

The fitness plot in Figure 4.7 shows that the evolutionary runs stagnated at around generation 60, and none of the runs were able to improve much after that.

Results from validation testing

All 256 uniform rules were tested 120 times against the same fitness-measure as was applied in the GA. The ReCA parameters were identical, and can be seen in Table 4.6. The

a_1	a_2	a_3	a_4	d_1	d_2	d_3	y_1	y_2	y_3	m_1	m_2
0	1	0	0	1	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0	0	1	0	0
1	0	0	0	1	1	0	0	0	1	0	0
0	1	0	0	1	0	1	0	0	1	0	0
0	1	0	0	1	1	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	1

Figure 4.6: Example data from the dual problem task. In addition to the 5-bit data, as seen in Figure 4.1, the system must also decide whether the additional three signals have a majority of 1's or 0's.

Table 4.11: GA runs performed on the 5-bit density task with $T_d = 10$.

Run no.	Gens.	Best fitn.	Best rule set	Validation fitn.
1	150	727	[17, 78, 90, 228, 160, 14]	583 ± 82
2	150	752	[14, 228, 56, 105, 39, 166]	568 ± 95
3	150	767	[60, 39, 56, 17, 180, 46]	497 ± 87
4	150	722	[102, 112, 72, 140, 180, 56]	579 ± 100
5	150	667	[18, 11, 153, 133, 36, 49]	638 ± 75
6	150	750	[17, 119, 60, 248, 81, 96]	572 ± 73
7	150	797	[60, 166, 84, 8, 80, 119]	607 ± 79
8	150	682	[209, 42, 179, 195, 101, 29]	559 ± 81
9	150	740	[92, 226, 195, 133, 46, 48]	631 ± 89
10	150	777	[166, 224, 153, 42, 174, 200]	570 ± 86

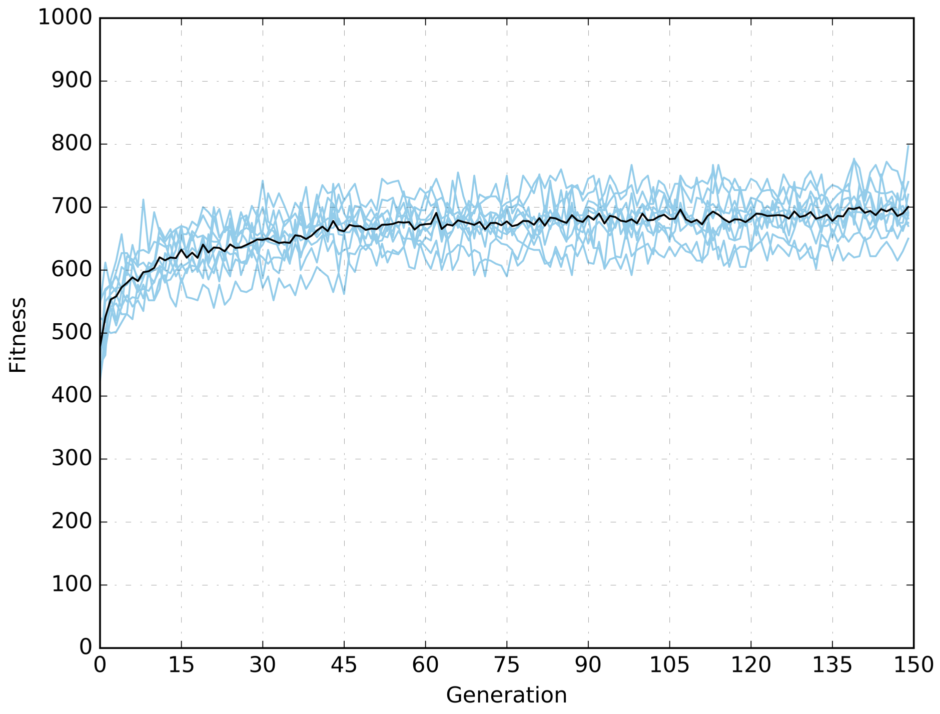


Figure 4.7: Fitness plot from GA-run to evolve a rule-set for the dual problem

Table 4.12: Fitness score of the tested rules on the dual problem. Only the best five best performing uniform rules are shown.

Rule	Wolfram Class	Fitness
Run 5	II & III & IV	638 ± 75
102	III	560 ± 65
14	II	558 ± 65
142	II	550 ± 65
174	II	549 ± 67
60	III	548 ± 61

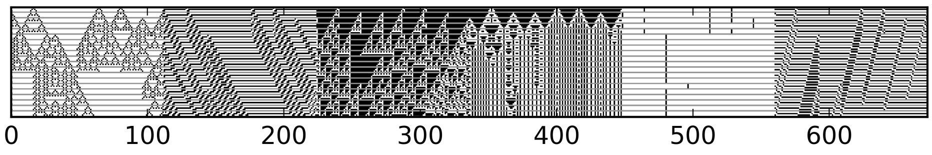
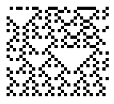
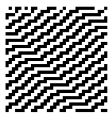
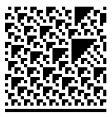
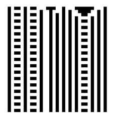
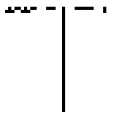
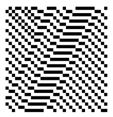
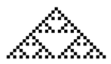
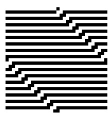
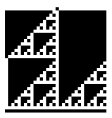
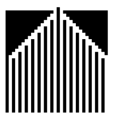
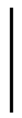
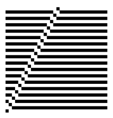


Figure 4.8: Example run of the evolved rule-set from GA Run 5 for the dual problem (5bit and density). The figure-style is explained in Figure 3.4.

Table 4.13: Rules evolved in Run no. 5, dual problem task with $T_d = 10$. The table shows which Wolfram class the rules belong to, a visualization of the rule with a random initial state, and a visualization of the rule evolved with single black cell in the middle.

Rule 18	Rule 11	Rule 153	Rule 133	Rule 36	Rule 49
W. cl. III	W. cl. II	W. cl. IV	W. cl. II	W. cl. II	W. cl. II
					
					

uniform rules have the permutation of mappings turned on, while the evolved rule-sets has it turned off. As seen in Table 4.12, the evolved rule from Run 5 was better than all the uniform rules, but not by a large margin. Note also that both Wolfram class II and III is present among the uniform rules.

Discussion

The evolved solution to the dual problem shows a larger degree of diversity in the rule set than what we have seen before. Wolfram classes II, III and IV are all represented. As seen in Figure 4.8, there seem to be complex behaviour in the first four rules. Rule number five (rule 36) seem to have a more role of indication. In rules two and six we see the same type of behaviour as in observed in the 20-bit problem.

Since the evolution stagnated it can be questioned if the CA was large enough. Increasing the parameters C or R could improve performance. However, this is equal for both the uniform rules and the quasi-uniform rule-set.

Table 4.14: ReCA parameters for the Japanese Vowels task. The B-parameter is new, and denotes how large vector each of the floats are mapped to. The usage of the parameters is described in Section 3.3.

N	52
R	8
C	12
I	1
B	4
Classifier	Perceptron SGD
Time-Transition function	Mapping copy
Permute mappings	No
Training examples	270
Testing examples	370

4.2 Sequence learning: Real world tasks

This section contains experiments that are designed to test both the research questions: **RQ1: Can quasi-uniform CA be a valuable addition to ReCA systems?**, and **RQ2: Can ReCA be used for sequence-to-sequence learning tasks?** The quasi-uniform rulesets will be evolved in the same manner as the previous section, but the length of the input-sequence and output-sequence may not be known.

4.2.1 Japanese Vowels

Problem description

The Japanese Vowels data-set was first presented in [26], and was later donated[1]. It consists of nine Japanese men uttering two Japanese vowels. The training set consists of the 30 utterances from each of the nine speakers, resulting in a total of 270 training examples. The test set consists of 370 examples, unevenly distributed between the nine speakers, ranging from 24 to 88 examples per speaker. The task is then to classify who of the nine speakers uttered the test-vowel.

The data of one utterance is divided into 12 floating point coefficients per time step, and total time steps per utterance ranging from 7 to 29.

The major difference from the previous examples is that the data in this experiment is represented as floating point numbers. The binarization scheme as described in Section 2.4.2 is used. All the floating points of the training data is analyzed and evenly divided into a number of intervals, given by the number B. 2^B gives the total number.

Evolving a solution

This task proved to be a much harder task then what was previously tested, mainly because of the large data-set. Since the input at each time-step consisted of 12 floating point numbers, and with the binarization scheme as described, the resulting size of the binarized

[1]https://archive.ics.uci.edu/ml/datasets/Japanese+Vowels, Accessed: 03.04.2017

Table 4.15: GA runs performed on the Japanese Vowels task.

Run	Gens.	Best fitn.	Best rule set	Val. testing fitn.
1	150	964	[199, 210, 248, 208, 112, 8, 15]	903 ± 41
2	150	978	[142, 12, 197, 127, 240, 55, 148, 169]	899 ± 48
3	150	967	[77, 15, 71, 188, 64, 157, 240, 98]	891 ± 38
4	150	967	[15, 158, 32, 208, 199, 233, 215, 168]	892 ± 104
5	150	967	[51, 108, 45, 166, 5, 197, 138, 4]	916 ± 64
6	150	964	[216, 124, 64, 132, 212, 192, 174]	912 ± 26
7	150	959	[207, 220, 232, 85, 157, 0, 159, 15]	912 ± 34
8	150	956	[142, 13, 66, 244, 137, 208, 45, 212]	918 ± 23
9	150	962	[158, 157, 138, 249, 85, 141, 166, 247]	911 ± 34
10	150	970	[212, 30, 23, 206, 174, 132, 141, 138]	908 ± 43

Table 4.16: Fitness score of the tested rules on the Japanese Vowels task. The rules were tested 60 times.

Rule	Wolfram Class	Fitness score
Run 8	II & III & IV	918 ± 23
138	II	907 ± 30
212	II	907 ± 31
142	II	906 ± 32
170	II	902 ± 23
240	III	901 ± 27

floats were $12 * 4 = 48$. Since the input data needed an eos-signal, an additional 4 bit were needed.

After a series of trail and error, the ReCA parameters as seen in Table 4.1 were used. The major difference from the previous experiments is that the I is set to just 1. A total of 10 evolutionary runs were done with the parameters in Table 4.2. None of the GA-runs managed to reach a fitness of 1000. The best performing rule-set was in Run 8, with a fitness after retesting of 918. This corresponds to a total of 30 misclassifications of the total 370 samples in the test-set. This is not quite state-of-the-art, as [22] managed to get zero misclassifications.

As seen in Figure 4.9, the GA-runs stagnated after about 50 generations.

Results from validation testing

All uniform rules were then tested on the task, and compared to the fitness of the best GA run. The ReCA parameters can be seen in Table 4.14. The results can be seen in Table 4.16. Since the Japanese Vowels task was the largest data-set so far, only 60 tests could be executed per rule. However, the standard deviation of the results are lower than previous experiments.

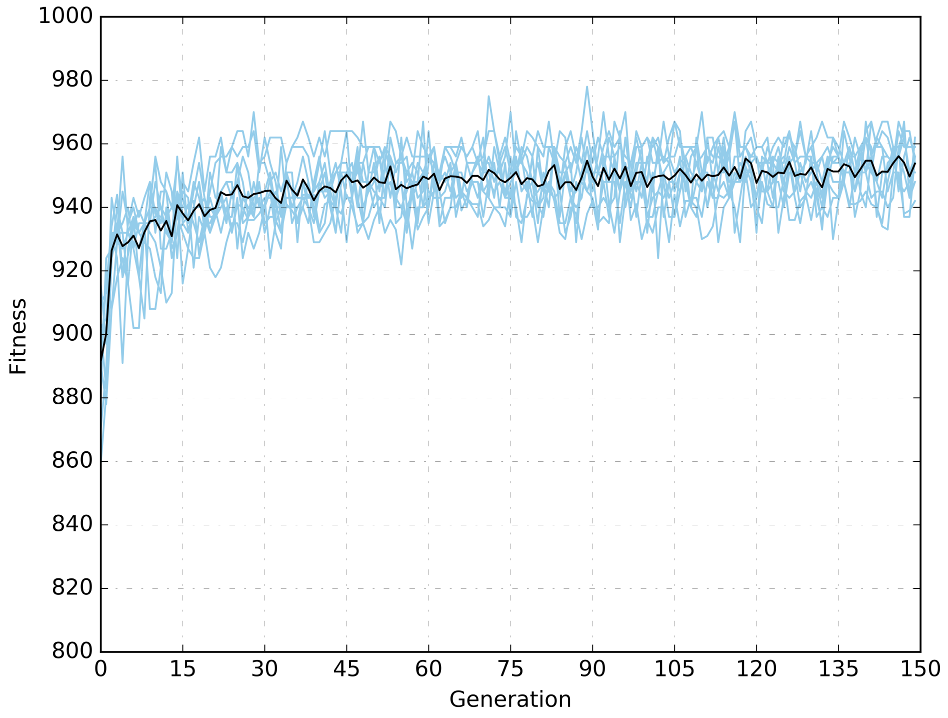


Figure 4.9: Fitness plot from the ten GA-runs to evolve a rule-set for the Japanese Vowels task. Note that the y-axis starts at 800.

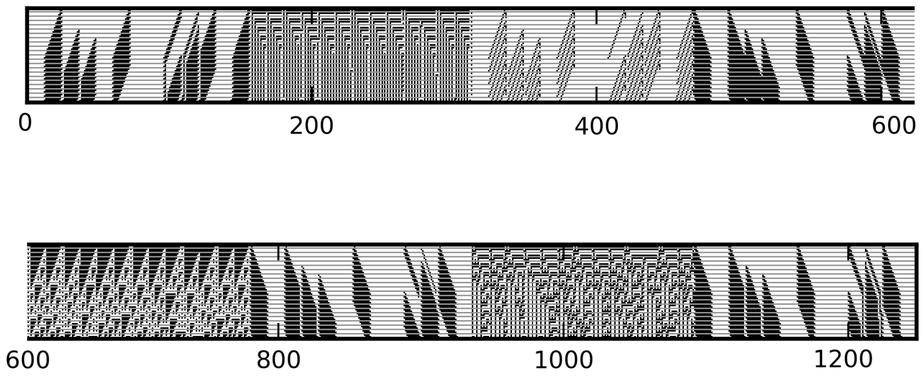
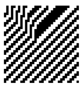
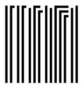
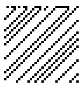
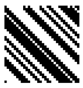
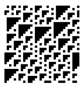
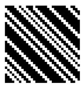
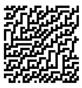
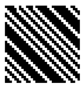
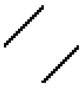
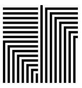
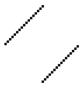
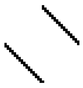
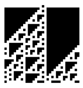
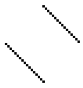
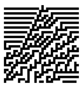
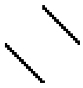


Figure 4.10: Example run of the evolved rule-set from GA Run 8 for the Japanese Vowels task. Note that the example visualization is done with $B = 1$ in order to reduce the size of the CA.The inter-rule dynamics remain unchanged. The CA is split in two in order to make it easier to read. The figure-style is explained in Figure 3.4.

Table 4.17: Rules evolved in Run no. 8, Japanese Vowels task. The table shows which Wolfram class the rules belong to, a visualization of the rule with a random initial state, and a visualization of the rule evolved with single black cell in the middle.

R. 142	R. 13	R. 66	R. 244	R. 137	R. 208	R. 45	R. 212
W.c. II	W.c. II	W.c. II	W.c. II	W.c. IV	W.c. II	W.c. III	W.c. II
							
							

Discussion

Once again the quasi-uniform rule-set did perform better than the uniform rules. However, the results were quite close, and the statistical significance of the results is not too large.

In Table 4.5 the rules of the best performing GA-run is presented. We see that Wolfram classes II, III and IV are present in the rule-set. As seen in figure 4.10, the rules does not seem to have a large degree of interactions. They do however project the input in different ways, perhaps making it easier for the classifier to extract important differences.

Because of the techniques introduced in Section 3.2.1 the different lengths of the utterances were not a problem. Jaeger et al. [22] use a snapshot technique for achieving the zero misclassifications, where the various sequence lengths are artificially padded to make them equally long. Perhaps this is needed to achieve state-of-the-art results.

4.2.2 Synthetic sequence-to-sequence problem

Problem description

In order to synthetically test the sequence-to-sequence capabilities of the ReCA system, a new task had to be made. The task is designed to test the systems capability of solving a problem that does not have an obvious correlation between the length of the inputs and output sequences. This new problem is named the Square Root Sequence problem.

The goal of the task is to determine how many 1's are present in a given temporal signal, and present this number as a sequence length, were the number of output time-steps must be the same number of the square root of the number of ones, floored to the nearest integer. Two examples of data for this task can be seen in Figure 4.11.

Evolving a solution

The ReCA parameters can be seen in Table 4.18. The task proved to be difficult. Table 4.19 shows that none of the ten GA runs reached 1000 in fitness.

As seen in Figure 4.12, all the ten GA runs stagnated around 60 generations.

x_1	x_2	x_3	eos	p	y_1	w	pes
1	1	1	0	0	0	1	0
0	1	0	0	0	0	1	0
0	1	1	0	0	0	1	0
1	1	0	0	0	0	1	0
0	0	0	1	0	0	1	0
0	0	0	0	1	1	0	0
		. . .			1	0	0
0	0	0	0	1	0	0	1

(a) Example 1

x_1	x_2	x_3	eos	p	y_1	w	pes
1	1	1	0	0	0	1	0
0	1	0	0	0	0	1	0
0	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	0	0	0	0	1	0
0	0	0	1	0	0	1	0
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
		. . .			1	0	0
0	0	0	0	1	0	0	1

(b) Example 2

Figure 4.11: Square Root Sequence problem data-set examples. The signals x_1, x_2, x_3 transmits an arbitrary number of 1's. The system is asked to produce the same number of y_1 outputs as the sum of the floored square root of the 1's in the input-signals. The data-type is explained in Section 3.2.1.

Table 4.18: ReCA parameters for the Square Root Sequence task. The usage of the parameters is described in Section 3.3.

N	5
R	6
C	12
I	2
Classifier	Perceptron SGD
Time-Transition function	Mapping copy
Permute mappings	No
Training examples	400
Testing examples	100
Sequence interval	(18-22)

Table 4.19: GA runs performed on the Square Root Sequence task.

Run	Gens.	Best fitn.	Best rule set	Validation testing fitness
1	150	820	[39, 71, 140, 93, 234, 77]	654 ± 144
2	150	815	[250, 80, 49, 96, 17, 181]	666 ± 100
3	150	835	[49, 227, 242, 17, 138, 175]	690 ± 110
4	150	820	[28, 77, 90, 165, 184, 3]	677 ± 87
5	150	845	[3, 238, 17, 123, 114, 39]	675 ± 125
6	150	845	[215, 240, 23, 59, 221, 3]	651 ± 120
7	150	835	[17, 232, 90, 5, 3, 87]	696 ± 85
8	150	835	[192, 39, 108, 194, 250]	650 ± 129
9	150	795	[120, 230, 115, 173, 250, 66]	622 ± 147
10	150	810	[35, 92, 241, 68, 195, 115]	643 ± 107

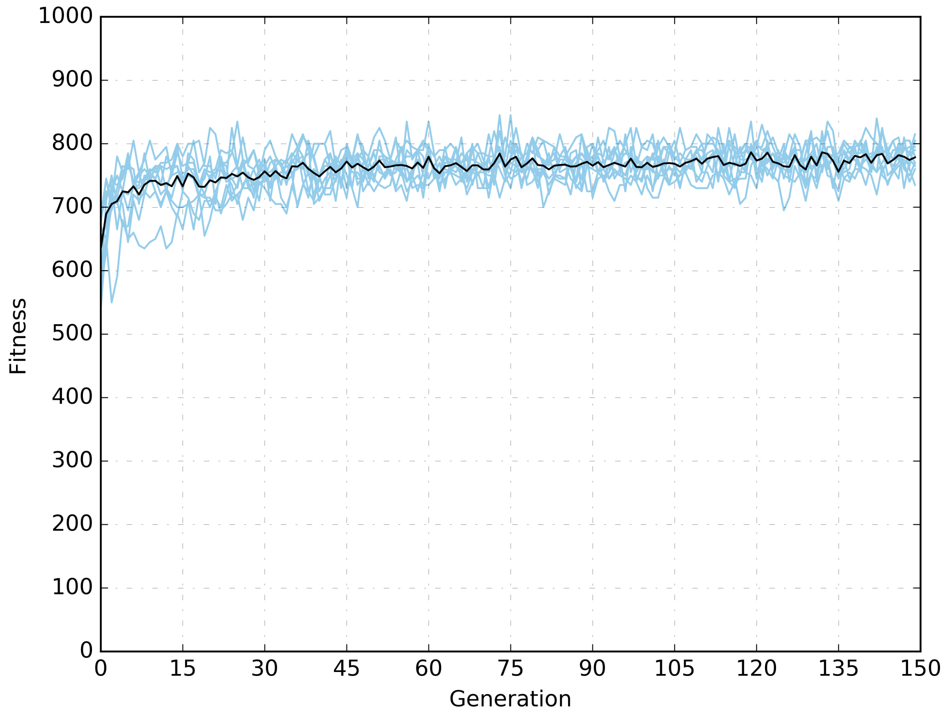


Figure 4.12: Fitness plot from the ten GA-runs to evolve a rule-set for the Square Root Sequence task.

Table 4.20: Fitness score of the tested rules on the Square Root Sequence task. The rules were tested 120 times.

Rule	Wolfram Class	Fitness score
Run 7	II & III	696 ± 85
17	II	568 ± 77
3	II	565 ± 65
115	II	529 ± 80
59	II	527 ± 81
35	III	505 ± 69

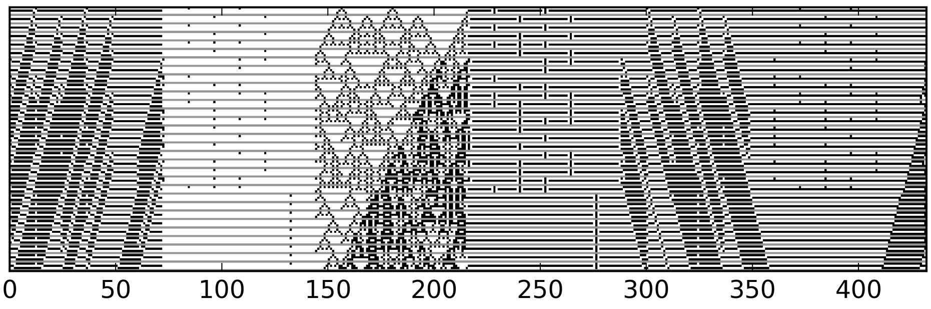


Figure 4.13: Example run of the evolved rule-set from GA Run 7 for the Square Root Sequence task. The figure-style is explained in Figure 3.4.

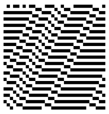
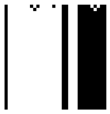
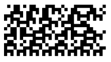
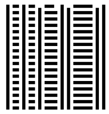
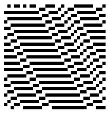
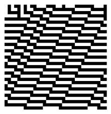
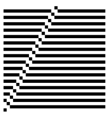
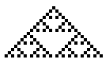
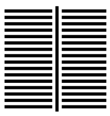
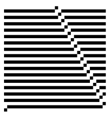
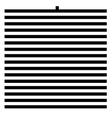
Results from validation testing

All 256 uniform rules were tested 120 times against the same fitness-measure as was applied in the GA. The ReCA parameters were the same, and can be seen in Table 4.18, with one exception. The uniform rules have the permutation of mappings turned on, while the evolved rule-sets has it turned off. As seen in Table 4.20, the Run 7 evolved rule-set outperformed the uniform rules. Again, the uniform rules are all in Wolfram class II.

Discussion

By investigating the space-time diagram in Figure 4.13 some interesting interactions are present. The static pattern on rule four is creating a downward-flowing triangle in the third rule. We can see similar interaction between rule one and two, where the static nature of rule two is creating a side-shifting pattern that stops right at the end of the input-sequence, where the prediction is starting. In Table 4.21 these interactions can be explained by the behaviour of the rules.

Table 4.21: Rules evolved in Run no. 7, 5-bit task with $T_d = 10$. The table shows which Wolfram class the rules belong to, a visualization of the rule with a random initial state, and a visualization of the rule evolved with single black cell in the middle.

Rule 17	Rule 232	Rule 90	Rule 5	Rule 3	Rule 87
W. cl. II	W. cl. II	W. cl. III	W. cl. II	W. cl. II	W. cl. II
					
					

a	b	c	_	eos	p	x	y	_	w	pes
1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	1	1	0	0	0	0
			. . .			1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	1

Figure 4.14: Example data for the translation task. The sentence aa_bb is translated to the sentence xyx_yyxx.

4.2.3 Machine translation: Europarl

This section includes the attempt on using ReCA for machine translation. It differs from the others because the problem is more difficult. It could therefore not be analyzed in the same way, but serves merely as an interesting usage of the ReCA system.

Problem description

The Europarl data-set contains the translated protocols from the European Parliament from English to many of the other languages in the European Union [25]. The translation problem is included in this thesis to show the usages of sequence-to-sequence learning, and new, practical usages of ReCA-systems on real-world tasks.

The translation problem is interesting because the mapping between a sentence in one language to another is not straightforward. Two sentences with the same semantics can be of vastly different lengths, and use different words that mean the same. This means that the length of both the input-sequence and the output-sequence is unknown.

Method for using ReCA for machine translation

The discrete nature of an alphabet is a perfect fit for a ReCA system. Each sentence is broken into a sequence of characters, in an attempt to do character-based translation [47]. The major advantage of this method is that the MT-system is alleviated of the challenged associated with prepossessing and tokenization [28].

Table 4.22: ReCA parameters for the machine translation task. The usage of the parameters is described in Section 3.3.

N	54
R	6
C	16
I	2
Classifier	Perceptron SGD
Time-Transition function	Mapping copy
Permute mappings	Yes
Training examples	8959
Testing examples	472
Sequence interval	(18-22)

In order to explain the method for using ReCA for machine translation, an example is given with a toy-problem. Language I has an alphabet of $A = \{a, b, c, \_\}$, where $\_$ is the white-space character, and a vocabulary of $V = \{aa, bb, abba\}$. Language II has an alphabet of $A = \{x, y, \_\}$, and a vocabulary of $V = \{xx, xyx, y\}$. An example translation is: $aa\_bb = xyx\_y\_xx$. This gives the character-by-character representation as seen in Figure 4.14.

In order to test the ReCA translation system on an actual task, the German-English translation set from the Europarl data-set was chosen. This meant that the alphabets were the Latin alphabet, the Arabic numbers, and some special character like white-space, comma and the period. For German, the letters *ä*, *ö* and *ü* were also allowed.

The total English-German corpus consists of 1.904.983 sentences. In order to reduce the data-set, all sentences of lengths between 18 and 22 were extracted from the corpus. In addition, some pre-processing was done; lower casing all letters, and removing all characters that did not exist in the alphabet. This results in the ReCA parameters as seen in Table 4.22.

Results

Unfortunately, the Europarl data-set is much too large to evolve a rule-set in the same way as the other tasks. However, the task presented in Section 4.2.2 have similarities with the translation problem. The length of the input-sequence is only loosely connected to the length of the output-sequence. This means that we can try the solutions found for the synthetic problem, on a real world task. Both the uniform rule 90 and the quasi-uniform rule-set from Run 7 on the Square Root Sequence problem in Section 4.2.2 were tested. Some example translations can be seen in Table 4.23.

Discussion

The rule-set evolved in Run 7 in Section 4.2.2 did not perform better than the uniform rule 90. Run 7 was not able to reproduce the concept of words, as no white-space was written. The rule 90 system was able to do this.

Table 4.23: Example translations. The English sentence is presented to the system, and the system then predicts. The $\longmapsto$-symbol indicates a forever repeating character.

English	Predict (Rule 90)	Predict (Run 7)	German label
are there any comments?	gibt es einwände?	gibseeses$\longmapsto$	gibt es einwände?
what does this mean?	wis bileute dee????	strrrrrrr	wie ist das gemeint?
it is as simple as that.	uasiiiifach i t auu	esesssss$\longmapsto$	so einfach ist da
that is the system.	aas sst e s eeenr.e.	eesssssss$\longmapsto$	das ist das system.

The transcripts from the European Parliament does not represent normal language usage. For example, the term ”are there any comments?” is frequently used, and is present both in the training set and the testing set. The rule 90 ReCA system was able to reproduce the correct translation, while the Run 7 system only was able to reproduce the first three letters correctly. The sentence ”that is the system” was not present in the training-set. Both rule 90 and Run 7 failed in producing anything meaningful as a translation. However, again the rule 90 system was able to produce words with white-spaces in between, and end the sentence with a period.

Chapter 5

Analysis

This chapter analyzes the outcome of the experiments, and the 50 evolutionary runs, both in relation to the ReCA system as a whole, and the usages of quasi-uniform CA, and sequence-to-sequence learning

5.1 On the ReCA system

The edge of chaos property has been regarded as an important aspect of the reservoir in RC. However, the best uniform rules in all five experiments are in class II. The classes of the rules in the evolved rule-sets are also mostly in class II. This is contrary to the results from previous publications [51, 30, 37, 31]. In order to explain this, the architecture and parameters of the ReCA system must be investigated.

The concept of using CA in RC is still new, and remains an active field of research. There are many architectural options, as described in Section 2.4. The effect of adjusting the parameters R ,C and I are somewhat unclear, and the difference between having an XOR Time-Transition function or the Mapping Copy style is also unclear.

The parameter I describes how many iterations the CA rule is applied to the bit-vector, and is the most important parameter for having the CA show emergent behaviour. However, increasing the I can lead to the information being destroyed, as reported by both [5] and [31]. In the experiments in this thesis, the I was held low; 2 for most experiments, and 1 for the Japanese Vowels. The low I could explain the unusual results with uniform rules in Wolfram class II. A higher I might yield the emergent behaviour of the CA rules. The stability issues of the results could also be connected with the parameter-selection. Since the research on ReCA still is somewhat premature, the precise effects of adjusting them are unknown.

Due to the stability issues, the ReCA system can not be regarded as a system that can be deployed on any operative task. In order to apply ReCA to a real-life problem, one must be certain that the system can perform above a critical level of performance.

Table 5.1: How many cells that each rule in the quasi-uniform rule-set included.

Problem	CA width($N * R * C$)	No. rules	Cells per rule
5-bit	$4 * 4 * 4 = 64$	4	16
20-bit	$7 * 6 * 16 = 672$	6	112
Dual problem	$7 * 6 * 16 = 672$	6	112
Japanese Vowels	$54 * 8 * 12 = 5184$	8	648
Sqrt. Seq. problem	$5 * 6 * 12 = 360$	6	60

5.2 Quasi-uniform CA in ReCA systems

A total of 50 evolutionary runs were made in order to investigate the selection of rule-set for use in quasi-uniform CA. In total, 34 out of 50 runs had at least one individual that was performing better than the best performing uniform rule for that experiment. All the top-performing rule-sets show a certain degree of inter-rule information flow. The common dynamic is the side-shifting of cells across different time-steps.

Why was the evolutionary run on the 5-bit problem so successful? The most important difference was that the GA was allowed to run for 1000 generations on the 5-bit task, but only 150 generations on the other tasks. The runs were restricted to 150 due to high computational demands. As seen in the 20-bit problem in Section 4.1.2, the evolution did not meet a local optima, and could possibly have evolved better solutions if ran longer.

By comparing to the results of the 5-bit problem, as seen in Table 4.3, only six of ten of the runs managed to get a fitness of 1000 before 150 generations were passed.

Another aspect about the 5-bit problem, is that it only has 32 possible training and testing examples, and that the same examples are used for both training and testing. This could have an effect on how the classifier is learning the behaviour of the reservoir.

Another difference from the well-performing 5-bit task to the others was the number of cells per rule, as seen in Table 5.1. The 5-bit problem only had 16 cell per rule, while the others ranged from 60 cell per rule, up to 648 cells per rule. This may have had an effect on the outcome. For 16 cells per rule on the 20-bit problem, a total of 42 rules would be needed. If the same rule is allowed to be chosen multiple times, this gives a total search space of $256^{42} \approx 1.4 * 10^{101}$ rule-combinations. In order to search through such a vast search-space, the efficiency of the algorithm must be improved.

As described in Section 3.4.2, each rule-set needed to be tested more than once, because the results were unstable. For most experiments in this thesis, each rule-set was tested four times in every generation. This was done to ensure that the search was guided towards rule-sets that actually were consistently well-performing, and not just randomly getting good results. If the stability of the results had been improved, it would not be necessary to test every rule-set so many times, and the speed of the GA could be drastically improved.

5.3 Sequence-to-sequence learning

Three of the experiments required the ReCA system to take an arbitrary length input-sequences. Two of them also needed an arbitrary output-sequence length. Did this framework function as intended? The largest selling point of the sequence-to-sequence system, was that it offered a convenient way of solving the problem of unknown lengths, on of both input and output sequences. However, this could cause implications. Since the length of sequences indeed vary, the size of the total ReCA reservoir also varies. This makes it difficult to adapt the size of the parameters to the total sequence length, and information might be lost in the reservoir dynamics.

Chapter 6

Conclusion

This section summarize the results from the work done in this thesis, and relates them to the research goal stated in the Introduction. Lastly, future work is outlined.

6.1 The usage of quasi-uniform CA in ReCA

The main research question of this thesis was the following: **Can quasi-uniform CA be a valuable addition to ReCA systems?**

Quasi-uniform CA has definitely proved to be a fruitful approach to ReCA-systems, and have given valuable insights in how CA reservoirs behave. The usage of an evolutionary process for finding the best combination of rules have also proved useful, but it has its limitations. On simple tasks, like the 5-bit problem, this approach proved very effective. However, on more difficult task, the gap between the uniform ReCA systems and the quasi-uniform ReCA system were not visible to the same degree.

Table 6.1 sums up the findings. The table gives a good indication of how the very best performing uniform rule is compared to the mean performance of the GA rule-sets.

However, the search proved to be computationally heavy, and was more difficult to use

Table 6.1: Quasi-uniform CA (Q-CA) vs. uniform CA (U-CA) on all problems. For each problem, the mean fitness of the best individuals over all ten GA runs is presented, along with the fitness-score of the best uniform rule. The two fitness-scores are compared by showing how much better the quasi-uniform fitness is.

Problem	Q-CA fitness	U-CA fitness	Q-CA/U-CA
5-bit	949	59	16.1
20-bit	491	918	0.535
Dual problem	580	560	1.03
Japanese Vowels	906	907	1.00
Sqrt. Seq. problem	662	568	1.17

on some tasks than others.

6.2 Sequence-to-sequence learning using ReCA

The second research question was the following: **Can ReCA be used for sequence-to-sequence learning tasks?** The sequence-to-sequence framework did work as intended, but was not able to produce state-of-the-art results. It did work as a proof-of-concept on the machine translation problem. However, the procedure was not able to produce state-of-the-art results on the known benchmark of Japanese Vowels. Perhaps the idea of the end-of-sequence token is a simplification of the real dynamics that is needed for these problems.

6.3 Contributions to the ReCA field of study

The research goal of this thesis was the following: **Doing novel research in the field of quasi-uniform ReCA systems, and the usage of ReCA in sequence-to-sequence learning.** The GA-procedure described in Section 3.4.2 show the ability of producing good results, and mostly better than selection of uniform rules. The results in Chapter 4 also show that it is not straightforward to chose which uniform rules are best, and some rules performed surprisingly well. The proposed GA provides a systematic way of a probabilistic outcome that is mostly better than the best uniform rule. The sequence-to-sequence framework presented in this thesis provides a novel method within ReCA of tackling the problem of unknown input and output sequence lengths. This proved to not being able to match the state-of-the-art on the Japanese Vowels data-set, but it has potential. The contributions made in this thesis is potentially a valuable addition to ReCA-research, and will hopefully motivate further research within the field.

6.4 Future work

6.4.1 More sophisticated evolutionary process

The focus of this thesis was a proof-of-concept behind the usage of quasi-uniform CA in ReCA systems. The GA that was used was a entry level GA. The genotype only consisted of a set of rules, and an upper limit of unique rules. The values of R (mappings), C (empty cells) and I (iterations) were all set by best judgement. Including these parameters in the genotype could allow the evolutionary process to decide their optimal values. However, the challenges of hyper-parameter selection is a well known issue within computer science, and the selection is probably not straightforward.

Also the non-domain specifics of the GA could be improved. Looking into more advanced genetic operators, multi-objective evolutionary algorithms and so on.

6.4.2 Towards truly non-uniform CA

The quasi-uniform CA that was used in this thesis only consisted of a small number of different rules. This could be expanded to a truly non-uniform CA rule-set, where each cell has its own rule.

6.4.3 Other types of CA

Only one dimensional elementary CA rules are explored in this thesis. When considering the various types of CA, the number of alternatives are near endless. Increasing the number of states, increasing the number of neighbors, and letting the CA-state take continuous values are some examples.

The most plausible next step however, is to increase the dimensionality of the CA. Much research has been done on two dimensional CA, and this might bring some new usages to ReCA. One problem with 1D CA is that information is spatially separated. Allowing higher dimension CA could alleviate this problem. Two dimensional CA is especially appealing for usage on image and video analysis. However, increasing the number of dimensions would introduce the well known curse of dimensionality.

6.4.4 Scaling up simulations with custom hardware or physical implementations.

A computational substrate as CA is vastly parallel, and not suitable for large scale simulation on a traditional computer, like the one the experiments in this thesis were conducted on. By using physical substrates for CA, like carbon nanotubes [11], or specialized hardware, like FPGAs [17], the speed of the simulation could be increased.

6.4.5 Full scale sequence-to-sequence learning

As demonstrated in Section 4.2.3, ReCA systems can be used for machine translation. However, when tested in this thesis, there were challenges with computational power, such that only sentences from length 18 to 22 could be used, constituting to about 0.49% of the total corpus of the English to German Europarl data-set.

Bibliography

[1] Bar-Yam, Y., 1997. Dynamics of complex systems. Vol. 213. Addison-Wesley Reading, MA.

[2] Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks 5 (2), 157–166.

[3] Bertschinger, N., Natschläger, T., 2004. Real-time computation at the edge of chaos in recurrent neural networks. Neural computation 16 (7), 1413–1436.

[4] Burkow, A. V., 2016. Exploring physical reservoir computing using random boolean networks. Master's thesis, NTNU.

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

[6] Conway, J., 1970. The game of life. Scientific American 223 (4), 4.

[7] Cook, M., 2004. Universality in elementary cellular automata. Complex systems 15 (1), 1–40.

[8] Darwin, C., 1859. On the origin of species.

[9] Dodd, M. S., Papineau, D., Grenne, T., Slack, J. F., Rittner, M., Pirajno, F., O'Neil, J., Little, C. T., 2017. Evidence for early life in earth's oldest hydrothermal vent precipitates. Nature 543 (7643), 60–64.

[10] Doursat, R., Sayama, H., Michel, O., 2013. A review of morphogenetic engineering. Natural Computing 12 (4), 517–535.

[11] Farstad, S. S., 2015. Evolving cellular automata in-materio. Master's thesis, NTNU.

[12] Fernando, C., Sojakka, S., 2003. Pattern recognition in a bucket. In: European Conference on Artificial Life. Springer, pp. 588–597.

[13] Fischer, I., Bueno, J., Brunner, D., Soriano, M. C., Mirasso, C., 2016. Photonic reservoir computing for ultra-fast information processing using semiconductor lasers. In: ECOC 2016; 42nd European Conference on Optical Communication; Proceedings of. VDE, pp. 1–3.

[14] Gershenson, C., 2004. Introduction to random boolean networks. arXiv preprint nlin/0408006.

[15] Gibbons, T. E., 2010. Unifying quality metrics for reservoir networks. In: Neural Networks (IJCNN), The 2010 International Joint Conference on. IEEE, pp. 1–7.

[16] Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, `http://www.deeplearningbook.org`.

[17] Halbach, M., Hoffmann, R., 2004. Implementing cellular automata in fpga logic. In: Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International. IEEE, p. 258.

[18] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y., 2016. Binarized neural networks. In: Advances in Neural Information Processing Systems. pp. 4107–4115.

[19] Hunter, J. D., 2007. Matplotlib: A 2d graphics environment. Computing In Science & Engineering 9 (3), 90–95.

[20] 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.

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

[22] Jaeger, H., Lukoševičius, M., Popovici, D., Siewert, U., 2007. Optimization and applications of echo state networks with leaky-integrator neurons. Neural networks 20 (3), 335–352.

[23] Jones, B., Stekel, D., Rowe, J., Fernando, C., 2007. Is there a liquid state machine in the bacterium escherichia coli? In: 2007 IEEE Symposium on Artificial Life. Ieee, pp. 187–191.

[24] Kleyko, D., Khan, S., Osipov, E., Yong, S.-P., 2017. Modality classification of medical images with distributed representations based on cellular automata reservoir computing. In: 2017 IEEE International Symposium on Biomedical Imaging.

[25] Koehn, P., 2005. Europarl: A parallel corpus for statistical machine translation. In: MT summit. Vol. 5. Citeseer, pp. 79–86.

[26] Kudo, M., Toyama, J., Shimbo, M., 1999. Multidimensional curve classification using passing-through regions. Pattern Recognition Letters 20 (11), 1103–1111.

[27] Langton, C. G., 1990. Computation at the edge of chaos: phase transitions and emergent computation. Physica D: Nonlinear Phenomena 42 (1), 12–37.

[28] Ling, W., Trancoso, I., Dyer, C., Black, A. W., 2015. Character-based neural machine translation. arXiv preprint arXiv:1511.04586.

[29] Lukoševičius, M., Jaeger, H., Schrauwen, B., 2012. Reservoir computing trends. KI-Künstliche Intelligenz 26 (4), 365–371.

[30] Margem, M., Yilmaz, O., 2016. An experimental study on cellular automata reservoir in pathological sequence learning tasks.

[31] McDonald, N., 2017. Reservoir computing and extreme learning machines using pairs of cellular automata rules. arXiv preprint arXiv:1703.05807.

[32] Miller, J. F., Downing, K., 2002. Evolution in materio: Looking beyond the silicon box. In: Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on. IEEE, pp. 167–176.

[33] Mitchell, M., 2000. Life and evolution in computers. History and philosophy of the life sciences 23 (3-4), 361–383.

[34] Mitchell, M., Crutchfield, J. P., Das, R., et al., 1996. Evolving cellular automata with genetic algorithms: A review of recent work. In: Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96). Moscow.

[35] Morse, G., Stanley, K. O., 2016. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference. ACM, pp. 477–484.

[36] Natschläger, T., Maass, W., Markram, H., 2002. The" liquid computer": A novel strategy for real-time computing on time series. Special issue on Foundations of Information Processing of TELEMATIK 8 (LNMC-ARTICLE-2002-005), 39–43.

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

[38] Nichele, S., Molund, A., 2017. Deep reservoir computing using cellular automata. arXiv preprint arXiv:1703.02806.

[39] 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. Journal of Machine Learning Research 12 (Oct), 2825–2830.

[40] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of go with deep neural networks and tree search. Nature 529 (7587), 484–489.

[41] Sipper, M., 1995. Quasi-uniform computation-universal cellular automata. Advances in Artificial Life, 544–554.

[42] Sipper, M., 1999. The emergence of cellular computing. Computer 32 (7), 18–26.

[43] Snyder, D., Goudarzi, A., Teuscher, C., 2013. Computational capabilities of random automata networks for reservoir computing. Physical Review E 87 (4), 042808.

[44] Sutskever, I., Vinyals, O., Le, Q. V., 2014. Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112.

[45] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–9.

[46] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2818–2826.

[47] Vilar, D., Peter, J.-T., Ney, H., 2007. Can we translate letters? In: Proceedings of the Second Workshop on Statistical Machine Translation. Association for Computational Linguistics, pp. 33–39.

[48] Von Neumann, J., Burks, A. W., et al., 1966. Theory of self-reproducing automata. IEEE Transactions on Neural Networks 5 (1), 3–14.

[49] Whitley, D., Rana, S., Dzubera, J., Mathias, K. E., 1996. Evaluating evolutionary algorithms. Artificial intelligence 85 (1), 245–276.

[50] Wolfram, S., 2002. A new kind of science. Wolfram media Champaign.

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

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