ANKARA YILDIRIM BEYAZIT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

CELLULAR AUTOMATA BASED RESERVOIR COMPUTING IN SEQUENCE LEARNING

Ph.D. Thesis by

Mrwan A. H. MARGEM

Department of Electrical and Computer Engineering

July, 2019

ANKARA

CELLULAR AUTOMATA BASED RESERVOIR COMPUTING IN SEQUENCE LEARNING

A Thesis Submitted to

The Graduate School of Natural and Applied Sciences of

Ankara Yıldırım Beyazıt University

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Electrical and Computer Engineering, Department of Electrical and Computer Engineering

by

Mrwan A. H. MARGEM

July, 2019

ANKARA

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"CELLULAR AUTOMATA BASED RESERVOIR COMPUTING IN SEQUENCE LEARNING"** completed by **MRWAN A. H. MARGEM** under the supervision of **ASSIST. PROF. DR. OSMAN S. GEDİK** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Ph.D.

Assist. Prof. Dr. Osman S. GEDİK

Supervisor

Associate Prof. Dr. Shafqat UR REHMAN

Jury Member

Associate Prof. Dr. Ömer KARAL

Jury Member

Assist. Prof. Dr. Elif VURAL

Jury Member

Assist. Prof. Dr. Javad RAHEBI

Jury Member

Prof. Dr. Ergün ERASLAN

Director

Graduate School of Natural and Applied Sciences

ETHICAL DECLARATION

I hereby declare that, in this thesis which has been prepared in accordance with the Thesis Writing Manual of Graduate School of Natural and Applied Sciences,

- All data, information and documents are obtained in the framework of academic and ethical rules,
- All information, documents and assessments are presented in accordance with scientific ethics and morals,
- All the materials that have been utilized are fully cited and referenced,
- No change has been made on the utilized materials,
- All the works presented are original,

and in any contrary case of above statements, I accept to renounce all my legal rights.

Date: **Signature:**

Name & Surname: Mrwan MARGEM

ACKNOWLEDGMENTS

Writing an acknowledgement... How can I fulfill this job without forgetting someone? So many interested people asked me how my PhD was progressing. I could not have succeeded without the help of so many, but I would like to give some special thanks.

Firstly, I would like to express my sincere gratitude to my supervisor, Assist. Prof. Dr. Osman S. GEDİK for his constant support, sharing the knowledge, mentoring, and his precious recommendations that assisted me all the time of my research and while writing this thesis.

I would like to express my appreciation and special thanks to my thesis committee: Associate Prof. Dr. Shafqat Ur REHMAN and Associate Prof. Dr. Ömer KARAL for their advices and the extended discussions during my thesis.

Special acknowledgement is given to the examiners of the thesis, Assist. Prof. Dr. Elif VURAL and Assist. Prof. Dr. Javad RAHEBI for agreeing to be the examiners of my PhD Viva.

My mother was there for me all the time, supporting and encouraging me all the way. I am really grateful to have such a loving and caring mother. My wife was critical to the success of this thesis, being there in good times and bad times, giving me strength to keep going. My children, thank you for your patience with me this long time.

Maybe also a small word of thanks to my own laptop which did not get tired despite the long trip.

Last, but not least, I thank my personal friends for their unconditional support and encouragement. This support means a lot to me.

2019, 18 July **Mrwan MARGEM**

CELLULAR AUTOMATA BASED RESERVOIR COMPUTING IN SEQUENCE LEARNING

ABSTRACT

Reservoir computing based on cellular automata (ReCA) constructs a novel bridge between automata computational theory and recurrent neural architectures. In this study, ReCA has been developed to solve different types of tasks. Several methods have been proposed to extract the features from the cellular automata reservoir. In most tasks, ReCA results outperform the state-of-the-art results.

Concerning the model complexity, a sparsely connected network with simple binary units like elementary cellular automata in ReCA could perform the computational requirements of the reservoir in order to solve hard sequence tasks that have long term dependencies. Thus, ReCA can be considered to operate around the lower bound of complexity.

Sequence learning is an essential capability for a wide collection of intelligence tasks such as language, continuous vision, symbolic manipulation in a knowledge base, etc. Therefore, ReCA has been tested using pathological synthetic tasks of sequence learning that are widely used in RNNs field. ReCA achieves zero error in all pathological tasks; using only the CA evolution states, at last time step, as a feature vector to predict the output (LAST method). The CA evolution states at all time steps (ALL method) can also be used, which improves the ReCA accuracy with large feature space. To reduce the feature space size, three options are proposed: Each by using only few states from the reservoir as features, Half by using only one side of CA evolution states, or *f* by reducing the dimension of the zero buffers. Using these three options together significantly reduces the ReCA complexity in some tasks by up to 98% for training and 94% for testing.

The distributed representation of CA in recurrent architecture (ReCA) could solve the 5-bit tasks with minimum complexity, using only *two* training examples which is the lowest number of training examples for any model. Comparing between different architectures and data representations; ReCA outperforms the local representation in

recurrent architecture (stack reservoir), then echo state networks and feed-forward architecture using local or distributed representation.

ReCA also could solve nonbinary tasks after using one hot encoding to binarize the dataset. The results are perfect for the signal classification and IRIS tasks where ReCA achieves zero error. While for the Japanese vowels task the results are competitive; less than the state-of-the-art results a little bit. Finally, ReCA has been tested using the 20 QA bAbI tasks from Facebook; These tasks are very hard and require an understanding of the meaning of a text and the ability to reason over relevant facts. Using only supporting facts, ReCA could solve most of bAbI tasks 15 out of 20 has 100% accuracy and 2 tasks above 90%, whilst 3 tasks less than 90%.

In addition, the usage of cellular automata in the reservoir computing paradigm greatly simplifies the architecture, makes the computation more transparent for analysis, and provide enough computation for large domain of tasks. Furthermore, the reservoir in ReCA can be implemented using ordinary logic gates or Field programmable gate arrays FPGAs, resulting in reducing the complexity in space, time and power consumption.

Thus, our work raises the question of whether real-valued neuron units are mandatory for solving complex problems that are distributed over time.

Keywords: Recurrent neural networks, reservoir computing, cellular automata, ReCA, sequence learning, long term dependencies, complexity, recurrent architecture, feed-forward architecture, distributed representation, local representation.

DİZİ ÖĞRENMESİNDE HÜCRESEL OTOMAT TEMELLİ REZERVUAR HESAPLAMA

ÖZ

Hücresel otomasyon temelli rezervuar hesaplama (ReCA) otomasyon hesaplama teorisi ve yinelemeli sinir ağları mimarileri arasında yenilikçi bir köprü kurmaktadır. Bu çalışmada, çeşitli tipteki görevlerin çözülmesi amacıyla ReCA önerilmiştir. Bununla birlikte, hücresel otomasyon rezervuarından çeşitli öznitelikleri çıkarmak için birçok yöntem önerilmiştir. ReCA birçok görevde tekniğin bilinen durumundan iyi performans göstermiştir.

Model karmaşıklığı göz önünde bulundurulduğunda, ReCA bünyesindeki temel hücresel otomasyon benzeri basit ikili birimler ile bağlı seyrek bağlanmış bir ağ, uzun süreli bağlılık içeren zor dizi görevlerin çözülebilmesi için gerekli hesaplama gereksinimlerini karşılayabilmektedir. Bu yönüyle ReCA'nın karmaşıklığın alt sınırlarında işlediği düşünülebilir.

Dizi öğrenmesi, dil işleme, sürekli görü işleme, bilgi tabanında sembolik manipülasyon gibi problemlerin çözülmesi için gerekli bir kabiliyet olarak karşımıza çıkmaktadır. Bu yönüyle ReCA, RNN literatüründe yaygın olarak kullanılan patolojik sentetik görevleri kullanarak test edilmiştir. ReCA sadece son adımdaki CA evrim durumlarını çıkışı tahmin etmek için öznitelik vektörü olarak kullanarak tüm patolojik görevlerde sıfır hataya ulaşmıştır (LAST yöntemi). Ayrıca tüm zaman adımlarındaki CA evrim durumları (ALL yöntemi) büyük öznitelik uzayı ile ReCA hassasiyetini arttırmak için kullanılabilmektedir. Öznitelik uzayı büyüklüğünü düşürmek için 3 yöntem önerilmiştir: EACH: Rezervuardaki sadece birkaç durumun kullanılması, HALF: CA evrim durumlarının sadece bir yarısının kullanılması veya *f*: sıfır tamponlarının boyutunun düşürülmesi. Bu üç yöntemin bir arada kullanılması ReCA karmaşıklığını bazı görevlerde eğitimde %98 ve testte %94 olmak üzere büyük oranda düşürmüştür.

ReCA mimarisinde CA'nın dağıtık gösterimi 5-bitlik görevleri sadece 2 eğitim örneği ile minimum karmaşıklık ile çözmekte olup bu durum herhangi bir model için en düşük

sayıda eğitim örneğidir. Farklı mimariler ve veri gösterimleri karşılaştırıldığında, ReCA'nın yinelemeli mimarilerdeki yerel temsiller (yığın rezervuarı) ile yerel ve dağıtık temsil kullanan yankı durum ağları ve ileri beslemeli mimarilerden iyi performans gösterdiği görülmüştür.

ReCA, veri seti bire bir kodlama ile ikili hale getirildiğinde ikili olmayan görevleri de çözebilmektedir. Bu şekilde gerçekleştirildiğinde sinyal sınıflandırma ve IRIS görevlerinde ReCA sıfır hataya ulaşarak mükemmel sonuçlar vermiştir. Öte yandan Japon ünlü harfleri görevindeki sonuçlar tekniğin bilinen durumunun bir miktar altında kalmıştır. Son olarak ReCA Facebook tarafından önerilen 20 QA bAbI görevinde test edilmiştir. Bu görevler oldukça zor olup, metnin anlaşılması ve ilgili gerçekler hakkında yorum yapılabilmesini gerektirmektedir. Sadece destekleyici gerçekler kullanılarak ReCA bu görevlerin çoğunu çözebilmiştir (15 görevde %100 doğruluk, 2 görevde %90 üzeri doğruluk ve 3 görevde %90 altı doğruluk)

Bununla birlikte, hücresel otomasyonun rezervuar hesaplamasındaki kullanımı mimariyi büyük ölçüde basitleştirerek hesaplamaların analiz amacıyla daha şeffaf olmasını sağlamakta ve büyük alan görevleri için yeterli hesaplama kabiliyeti sunmaktadır. Ayrıca, ReCA bünyesindeki rezervuar, basit mantık kapıları ve FPGA ile de gerçeklenerek ebat, zaman ve güç tüketimi karmaşıklıklarını azaltabilmektedir.

Böylelikle bu çalışma zaman içerisine yayılmış kompleks problemlerin çözülebilmesi için reel-değerli nöronların gerekliliğinin sorgulanmasını sağlamaktadır.

Anahtar kelimeler: Yinelemeli sinir ağları, rezervuar hesaplama, hücresel otomasyon, ReCA, dizi öğrenmesi, uzun süreli bağlılık, karmaşıklık, yinelemeli mimari, ileri beslemeli mimari, dağıtık temsil, yerel temsil

CONTENTS

NOMENCLATURE

Roman Letter Symbols

$\mathbf{A}$	Combination of $A^{(n)}$ for all time steps, i.e., n changes from 1 to T
$A^{(ALL)}$	Vector concatenation of $\mathbf{A}$
$A_i^{(n)}$	CA evolution state at iteration i in time step n
$A^{(n)}$	Concatenation of all iterations (from 1 to I) of CA evolution states in time step n
F	Reservoir activation function in ESNs
f	Expansion ratio
I	Total number of CA iterations in the reservoir at single time step
k	Number of selected states in Each option that are used as feature vectors
L	Dimension of X_n
L_{CA}	Dimension of feature vector
L_{in}	Dimension of $\mathbf{u}(n)$
L_{out}	Dimension of $\mathbf{y}(n)$
N	Number of reservoir neurons (Reservoir size)
Nn	Number of Neighbors
Ns	Number of States
N_{test}	Number of testing examples
N_{train}	Number of training examples
r	CA radius
R	Number of input permutation or the dimension of the zero buffer.
R_i	Number of bits to represent a single bit of input to reduce the interference.
S	CA state set
T	Number of time steps in the dataset (Sequence Length)
T_d	Distractor period
$\mathbf{u}(n)$	Original Input at time step n
$\mathbf{W}$	Reservoir weight matrix in ESNs
$\mathbf{W}_{in}$	Input Weight Matrix in ESNs
$\mathbf{W}_{out}$	Output Weight Matrix
$\mathbf{x}(n)$	Reservoir state vector at time step n in ESNs.
X_n	Input at time step n after preprocessing before the reservoir
$\boldsymbol{X_{train}}$	State collection matrix from all training examples

$(\boldsymbol{X_{train}})^{\dagger}$	Pseudo-inverse of $\boldsymbol{X_{train}}$
$\mathbf{Y}$	Output matrix
$\mathbf{y}_{(b)}$	Binarized output
$\mathbf{y}(n)$	Output at time step n
$\hat{\mathbf{y}}(n)$	Predicted output
$\boldsymbol{Y_{train}}$	Collection output matrix from all training examples (the target)

Greek Letter Symbols

α	Leaking rate in ESNs
ρ(**W**)	Spectral radius of **W** in ESNs
σ	Input weight scaling in ESNs

Subscripts

b	Binarized
d	Distractor
i	Iteration number
in	Input
out	Output
n	Time step
train	Training examples
test	Testing examples

Acronyms

AI	Artificial Intelligence
ALL	Using all iterations of CA evolution states in all time steps as a feature space.
ANNs	Artificial Neural Networks
BPDC	Back-Propagation Decorrelation Neural Network
CA	Cellular Automata
CAr	Local Rule of Cellular Automata
DL	Deep Learning
Each	Using only k evolution states for each time step as a feature space
ECA	Elementary Cellular Automata

ECAM	Elementary Cellular Automata with Memory
ELMs	Extreme Learning Machines
ESNs	Echo State Networks
FNNs	Feedforward Neural Networks
FPGAs	Field Programmable Gate Arrays
GPU	Graphics Processing Unit
Half	Using only one side of CA evolution states as a feature space
LAST	Using all iterations of CA evolution states in last time step as a feature space.
LPC	Linear Predictive Coding
LRi	Linear Regression Input
LSB	Least Significant Bit
LSMs	Liquid State Machines
LSTM	Long Short-Term Memory
MemN2N	End-to-End Memory Networks
MemNN	Memory Networks
ML	Machine Learning
MSB	Most Significant Bit
NLP	Natural Language Processing
NMSE	Normalized Mean Square Error
NMT	Neural Machine Translation
NR	Neural Reasoner
NS	Next State
NTM	Neural Turing Machine
PS	Present State
QA	Question Answering
RC	Reservoir Computing
RCNs	Reservoir Computing Networks
ReCA	Reservoir Computing based on Cellular Automata
RNNs	Recurrent Neural Networks
SF	Supporting Facts
SVMs	Supporting Vector Machines
TPR	Tensor Product Representation
var(**y**)	Output variance

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Artificial neural networks (ANNs) are powerful tools for artificial intelligence (AI) and machine learning (ML), which have remarkable progress in recent years, and they are increasingly employed in real-life applications [1]. ANNs are computational models that mimic biological neural networks. They are represented by a network of units (neurons) interconnected via weighted links (synapses). The ANN architectures can be classified into two classes from the point of view of the connection direction: feedforward neural networks (FNNs) and recurrent neural networks (RNNs) [2]. In FNNs there are only one-way connections (from input to output), which makes it suitable for static data processing and representing the nonlinear input-output functions. But, in RNNs, the feedback connections are allowed; thus, they can represent dynamical systems driven by sequential inputs owing to their feedback connections, that make them powerful tools for dynamic (time-dependent) data processing [2]. Unfortunately, RNNs are very difficult to train by traditional methods as gradient descent [3]. To overcome this difficulty; RNN is divided into two networks: a non-trainable RNN (fixed dynamic reservoir) and trainable feedforward ANN (read-out). Thus, the problem changes from RNN training to feedforward ANN training which is very common in ML, but instead very often becomes a simple linear readout as in echo state networks (ESNs) [4]. This method of simplification became known as reservoir computing (RC). ESNs [4], Liquid State Machines (LSMs) [5] and the back-propagation decorrelation neural network (BPDC) [6] are some popular examples for RC models. The high dimensional projection can be provided by cellular automata (CA) evolution states instead of the RNN reservoir as in [7], which greatly simplifies the architecture complexity and makes the computation faster and more transparent for analysis [8, 9].

The following subsections are a review of reservoir computing, cellular automata, and overview of reservoir computing based on cellular automata, then our contributions, finally the thesis structure.

1.1 Reservoir Computing

RNNs are connectionist computational systems where they can embed temporal correlations of the inputs into their dynamical behavior, which makes them suitable to solve time-dependent tasks (problems) such as speech recognition, language modeling, financial data analysis, etc.

RNNs are universal approximators of dynamical systems [10] and can simulate Turing machines [11], but it is very difficult to train them, due to convergence problems [12] and the difficulties of finding optimal representations for long-term memory learning [3, 13]. Reservoir Computing is an approach that could avoid these difficulties by splitting the network into two layers, as defined above: the first one is a non-trainable (fixed) layer called (a reservoir) whilst the second layer is a trainable feedforward FNNs (a readout layer). Hence, the sequence input is projected into spatiotemporal patterns in a high-dimensional space by the RNN reservoir. Then, the features are extracted in the readout, as shown in Figure 1.1(a). The main advantage of RC framework is that the output matrix weights $\mathbf{W}_{\text{out}}$ are only trained, but the input matrix weights $\mathbf{W}_{\text{in}}$ and the recurrent connection weights $\mathbf{W}$ in the reservoir are not trained. This simple and fast training process overcomes the issues in [3, 12, 13] and further reduces the computational cost of learning compared with standard RNNs. The RC models differ in the reservoir construction and the read-out type. For example, ESNs [4] that have randomly and sparsely connected recurrent neurons in the reservoir and linear read-out. In LSMs [5], the reservoirs are driven by the dynamics of a set of coupled spiking integrate-and-fire neuron models and they mostly have linear read-out (some cases have FNNs or sigmoid neurons) [14]. BPDC is an online RNN learning algorithm using the algorithm of Atiya and Parlos in [15] by adapting only the output weights. Other types of reservoirs can be found in [16-23].

In this thesis, ESNs have been adopted due to their simplicity and effectiveness, also they have been successfully applied in multiple tasks, e.g., dynamic pattern classification [24], time-series prediction and noise modelling [25], reinforcement learning [18], speech recognition [26], language modelling [27], human action

recognition [28], finally handwriting recognition and movement detection [29]. In [29] the authors call ESNs by reservoir computing networks (RCNs).

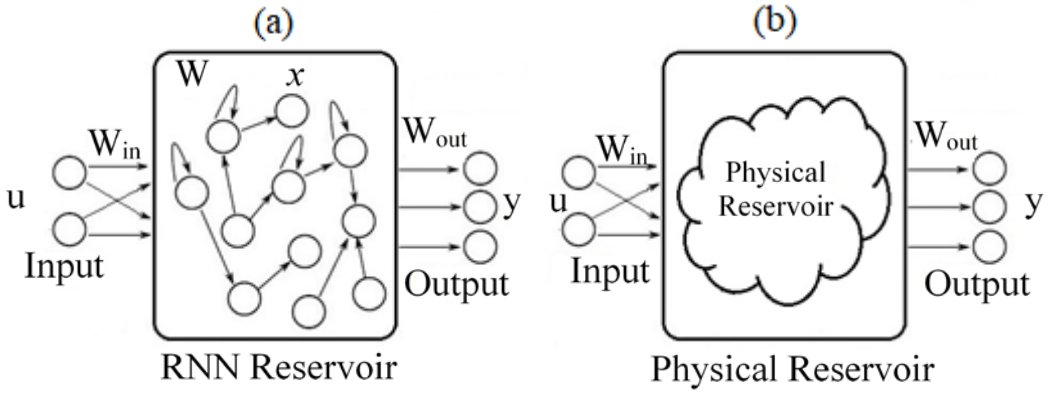


Figure 1.1 Reservoir computing paradigm: (a) The input is projected randomly in an *RNN* (reservoir) and the evolution states of the reservoir x(n) are harvested to calculate the output weight matrix $\boldsymbol{W}_{out}$ which is then used to predict the output. (b) Physical Reservoir: The RNN reservoir in (a) is replaced by a high dimensional dynamic physical system.

1.1.1 ESNs Framework

In the RC model shown in Figure 1.1(a), and for more precise ESNs with mathematically speaking; given K examples for training, the target is to find M-dimensional output $\mathbf{y}$(n) as a function of L-dimensional input $\mathbf{u}$(n), where n is the time step number n=1, 2, …, T where T is the input sequence length.

The update equation of the reservoir states is:

$$\boldsymbol{x}(n) = F\big(\boldsymbol{W}.\boldsymbol{x}(n-1) + \boldsymbol{W_{in}}.\boldsymbol{u}(n)\big) \tag{1.1}$$

where, $\boldsymbol{x(n)}$ is the reservoir state vector at time step n, $\mathbf{x}(0) = \mathbf{0}$, F is the reservoir activation function, $\mathbf{W}_{\text{in}}$ is an N×L matrix of random projection of the input to the reservoir, $\mathbf{W}$ is an N×N matrix which describes the internal connections and weights of the reservoir neurons and N is the reservoir size (the number of reservoir neurons).

In order to efficiently solve computational tasks, the reservoir should verify the following requirements:

1- *High dimensionality*; to map inputs into a high-dimensional space.
2- *Nonlinearity*; to transform not linearly separable inputs into those that are linearly separable.
3- *Echo state property* or fading memory; the network should gradually lose information that has been received from previous states and inputs [5, 30]. To ensure the *echo state property* in the reservoir, the *spectral radius* $\rho(\boldsymbol{W})$[1] of the reservoir matrix should be less than 1 [31], but in [32] it was proved that the spectral radius $\rho(\boldsymbol{W})$ still has to be found by task-specific experimentation.
4- *Edge of chaos*; it is often preferred to operate *ESN*s at the edge of chaos [33,34].

Due to the rich dynamics in the reservoir, the output can be simply expressed as the weighted sum of the reservoir states as in equation (1.2):

$$\boldsymbol{y}(n) = \boldsymbol{W_{out}}.\boldsymbol{x}(n) \qquad (\mathbf{1.2})$$

where $\boldsymbol{W}_{out}$ is the $M \times N$-dimensional output weight matrix which is only trained whilst $\boldsymbol{W}_{in}$ and $\boldsymbol{W}$ are created randomly and are not changed during training. However, in traditional RNNs, the training methods adapt all the weights ($\boldsymbol{W}_{in}$, $\boldsymbol{W}$, $\boldsymbol{W}_{out}$).

In ESNs, $\boldsymbol{W}_{out}$ can be obtained directly in training stage from equation (1.3):

$$\boldsymbol{W_{out}} = \boldsymbol{Y_{train}}(\boldsymbol{X_{train}})^{\dagger} \qquad (\mathbf{1.3})$$

where $(\boldsymbol{X_{train}})^{\dagger}$ is the pseudo-inverse of $\boldsymbol{X_{train}}$ which is the state collection matrix from all training examples and $\boldsymbol{Y_{train}}$ is the collection output matrix from all training examples (the target). Thus, the most expensive calculation in the training stage of the RC model is to find the pseudo-inverse of $x(n)$.

Finally, to predict new output values $\hat{\mathbf{y}}$ other than the output $\boldsymbol{y}$ of the K training examples that have been used to find the output weight matrix $\boldsymbol{W}_{out}$; equation (1.4) is used as follows:

[1] The *spectral radius* $\rho(\boldsymbol{W})$ *is* the maximum eigenvalue of the reservoir weight matrix $\boldsymbol{W}$.

$$\hat{\boldsymbol{y}}(n) = \boldsymbol{W_{out}}\hat{\boldsymbol{x}}(n) \quad (\mathbf{1.4})$$

where $\hat{\boldsymbol{x}}(n)$ *is* the new reservoir states that are obtained from the new input. For more details, Lukosevicius in [35] presents practical techniques and recommendations for applying ESNs.

1.1.2 Physical Reservoir

Some of high dimensional dynamic physical systems can be considered as a computational model [36], which led the researchers to exploit these systems as physical reservoirs (Figure 1.1(b)). The physical reservoir, of course, should verify the requirements of a conventional reservoir that are mentioned in Section 1.1.1. Using physical systems as a reservoir starts in 2003 by water waves in a bucket [37], then the genetic regulatory network of the *Escherichia coli* bacterium [38, 39], optoelectronics [40–42], random Boolean networks (RBNs) [43], carbon nanotubes [44, 45], coupled oscillators; chemical [46], phase [47] and mechanical [48], finally using CA as a reservoir [7-9, 49-51]. Other types of physical reservoirs can be found in the review paper [52]. Also, there are many unconventional computing methods in [53] can be used as physical reservoirs.

1.2 Cellular Automata

Cellular automata (CA) have been originally studied by J. von Neuman in the 1950s as a mathematical model for the self-reproducing phenomena [54-56]. CA have since been used to simulate a wide variety of physical and biological systems [57-59]. They are also useful for parallel computation and can be easily implemented using hardware such as field programmable gate arrays (FPGAs) [60] and graphics processing unit (GPU) [61].

CA are composed of simple computational cells arranged on a lattice to produce a network, which can be evolved to extremely complicated behavior[2], thus CA can

[2] CA have emergent properties; because their global behavior arises from the complex interactions between their cells; is not designed into these cells [62].

perform larger computational tasks. Precisely, CA are a discrete (in time and space) parallel computational model composing of a lattice of cells. The cell state evolves in time according to a certain transition function (rule), depending on its current state and the state of neighbors [63]. The states of all cells in the array are updated simultaneously, each update is called iteration and the total number of iterations is denoted by I.

1.2.1 Dimension and Neighborhood

The original CA, that have been proposed by J. von Neumann [54], is of 2-dimensional with 5-neighborhood dependency (orthogonal ones and itself) as shown in Figure **1.2**(a). In 1962, the neighborhood dependency has been extended by Moore [64] to the 9-neighborhood dependency as shown in Figure 1.2(b), the Moore neighborhood structure has been utilized to design the famous Game of Life by John Conway and popularized by Martin Gardner (1970) [65].

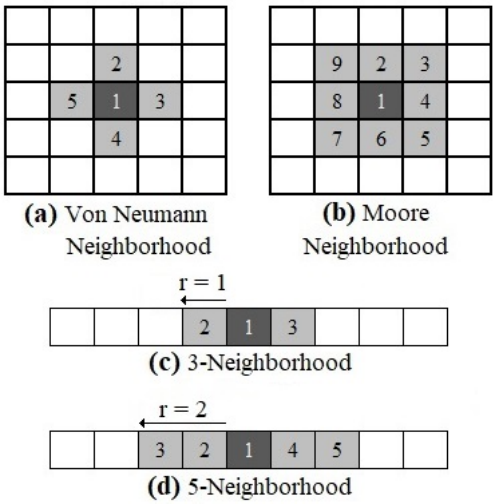


Figure 1.2 Neighborhood Dependency: (a) 5-Neighbors, (b) 9-Neighbors, (c) 3-Neighbors, and (d) 5-Neighbors.

Figure 1.2(c,d) describes the 1-dimensional CA with 3-neighborhood (Figure 1.2(c)) and 5-neighborhood (Figure 1.2(d)). The neighborhood is sometimes represented by *radius r*, i.e., the number of consecutive cells in a direction on which the dependency, e.g., *r*=1 in Figure 1.2(c) and, *r*=2 in Figure 1.2(d). In 1-dimensional CA, the number

of neighborhood dependency Nn=2*r*+1. The radius is usually the same in every direction, but this restriction is removed in some literature as [66-68].

1.2.2 Cell States

A CA cell can be in any state *s* of a finite state set *S*. The simplest representation of the state set S is the binary, i.e., $S \in \{0,1\}$ and the number of states Ns=2, which has been used in the 2-dimensional Conway's game of life. The cell is a*live* and is normally represented by a black square, and as opposed, when s=0 the cell is *died* and is normally represented by a white square. The 1-dimension CA with binary state set with three dependent neighbors is called elementary cellular automata (ECA), which is widely used in CA discipline; due to its simplicity. If the state set *S* consists of *three* numbers {0, 1, 2} and the new state is the average of its previous state and the *two* neighbors, CA will be called *totalistic* CA [63]. Finally, if the state set *S* is any real number, CA will be called *continuous* CA [63].

1.2.3 Local Rule

A cell (in present state PS) of CA changes its state (to next state NS) after applying a transition function (rule) on its previous state and the previous state of the dependent neighbors.

Table 1.1 ECA rules: PS is the present state of the 3-neighbors, NS is the next state (update) of the center cell, the last column is the rule number, which is the decimal equivalent of its 8-bit binary string.

PS	111	110	101	100	011	010	001	000	Rule #
NS	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	1	1
	:	:	:	:	:	:	:	:	:
	0	1	0	1	1	0	1	0	90
	:	:	:	:	:	:	:	:	:
	0	1	1	0	1	1	1	0	110
	1	1	0	0	1	0	1	1	203
	:	:	:	:	:	:	:	:	:
	1	1	1	1	1	1	1	0	254
	1	1	1	1	1	1	1	1	255

The local rule can be represented in a tabular form as demonstrated in Table 1.1 that contains the next state values corresponding to each of the possible neighborhood

combinations according to the local rule [69]. ECA rules have got their numbers (or names) after the decimal equivalents (last column of Table 1.1) of their 8-bit binary string. Obviously, Table 1.1 shows there are 256 different ECA rules from rule 0 to rule 255, where it can be mathematically expressed as follows: for ECA rule space there are $(Ns)^{(Ns)^{Nn}}=2^{2^3}=256$ different rules [70].

Table 1.2 88 equivalent sets of ECA rules with their Wolfram classes.

Rule	Class	Equivalent Rules	Rule	Class	Equivalent Rules	Rule	Class	Equivalent Rules
0	I	255	35	II	49, 59, 115	106	IV	120, 169, 225
1	II	127	36	II	219	108	II	201
2	II	16, 191, 247	37	II	91	110	IV	124, 137, 193
3	II	17, 63, 119	38	II	52, 155, 211	122	III	161
4	II	223	40	I	96, 235, 249	126	III	129
5	II	95	41	IV	97, 107, 121	128	I	254
6	II	20, 159, 215	42	II	112, 171, 241	130	II	144, 190, 246
7	II	21, 31, 87	43	II	113	132	II	222
8	I	64, 239, 253	44	II	100, 203, 217	134	II	148, 158, 214
9	II	65, 111, 125	45	III	75, 89, 101	136	I	192, 238, 252
10	II	80, 175, 245	46	II	116, 139, 209	138	II	174, 208, 224
11	II	47, 81, 117	50	II	179	140	II	196, 206, 220
12	II	68, 207, 221	51	II	-	142	II	212
13	II	69, 79, 93	54	IV	147	146	III	182
14	II	84, 143, 213	56	II	98, 185, 227	150	III	-
15	II	85	57	II	99	152	II	188, 194, 230
18	III	183	58	II	114, 163, 177	154	II	166, 180, 210
19	II	55	60	III	102, 153, 195	156	II	198
22	III	151	62	II	118, 131, 145	160	I	250
23	II	-	72	II	237	162	II	176, 186, 242
24	II	66, 189, 231	73	II	109	164	II	218
25	II	61, 67, 103	74	II	88, 173, 229	168	I	224, 234, 248
26	II	82, 167, 181	76	II	205	170	II	240
27	II	39, 53, 83	77	II	-	172	II	202, 216, 228
28	II	70, 157, 199	78	II	92, 141, 197	178	II	-
29	II	71	90	III	165	184	II	226
30	III	86, 135, 149	94	II	133	200	II	236
32	I	251	104	II	233	204	II	-
33	II	123	105	III	-	232	II	-
34	II	48, 187, 243						

These 256 rules have a lot of equivalences. Consequently, a similar behavior can be classified in a set of rules, thus ECA rule-space can be reduced to 88 equivalent sets of rules as listed in Table 1.2 [71]. But, the most commonly used is the visual representation where the zero is represented by a white square and the one by a black square [63] as illustrated in Figure 1.3 for rule 150 as an example.

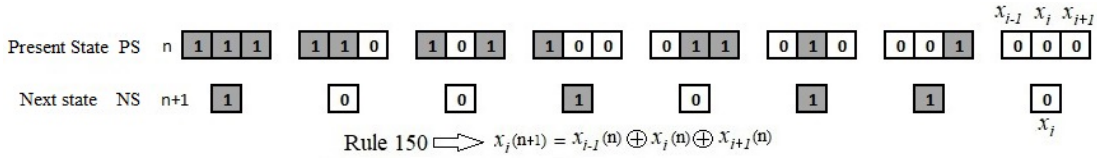


Figure 1.3 ECA Rule 150: All possible combination for one iteration of the center cell x_i for rule 150 and its Boolean expression. The binary string $(10010110)_2$ produces the rule number $(150)_{10}$ [85].

1.2.4 Boundary Condition

In 1-dimensional finite CA, two boundary conditions are generally used; *fixed* and *periodic* boundary conditions.

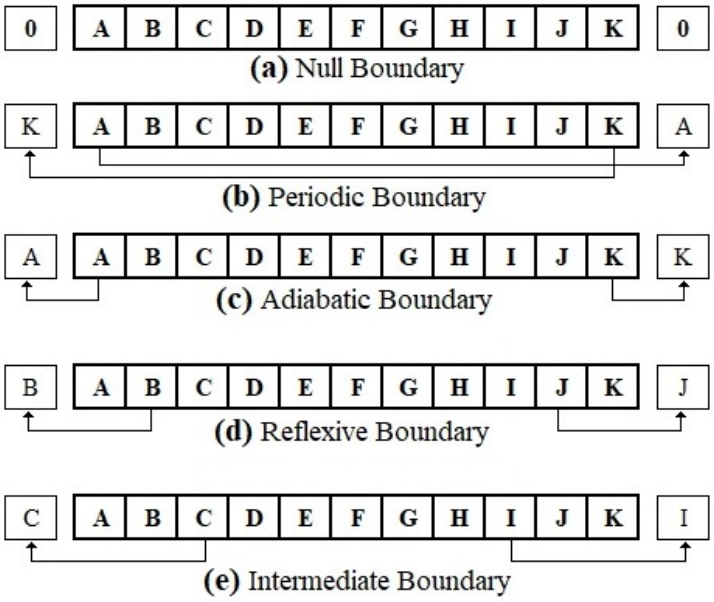


Figure 1.4 Boundary Conditions: (a) Null Boundary, (b) Periodic Boundary, (c) Adiabatic Boundary, (d) Reflexive Boundary, (e) Intermediate Boundary.

In fixed boundary condition; the rightmost and the leftmost neighbors are in a fixed value, it is also possible to take different boundary values between right and left terminal as in [72]. The most popular boundary condition in fixed type is *null* boundary, where the rightmost and the leftmost neighbors are always in state 0 as shown in Figure 1.4(a). In periodic boundary condition, the grid is folded, i.e., the leftmost cell becomes the neighbor of the rightmost one and vice versa *as* described in Figure 1.4(b). There are other types of boundary conditions that fall within the periodic

type; depending on the position of the selected boundary cell as demonstrated in Figure 1.4(c, d, e) [73].

1.2.5 Elementary Cellular Automata Classification

From here, ECA will only be used; due to its simplicity, but the other types of CA can be used in future work to exploit the other specifications rather than ECA. Starting by a random initial vector, the evolution of ECA states obeys different behaviors, that can be categorized into 4 classes (Wolfram classes) [63]:

Class I (*Uniform*): the initial states evolve to a stable (constant) state, e.g., rule 235, 255 all the cells become *1*s and rule 0, 128 all the cells become *0*s (Figure 1.5(a));

Class II (*Periodic*): CA states evolve into static or oscillating states, e.g., rule 42, 108 (Figure 1.5(b));

Class III (*Chaotic*): CA states evolve chaotically, e.g., rule 90, 105, 150 (Figure 1.5(c));

Class IV (*Complex* or *Edge of Chaos*): the initial vector evolves in an unpredictable manner (complex behaviors), e.g., rule 106, 110 (Figure 1.5(d)).

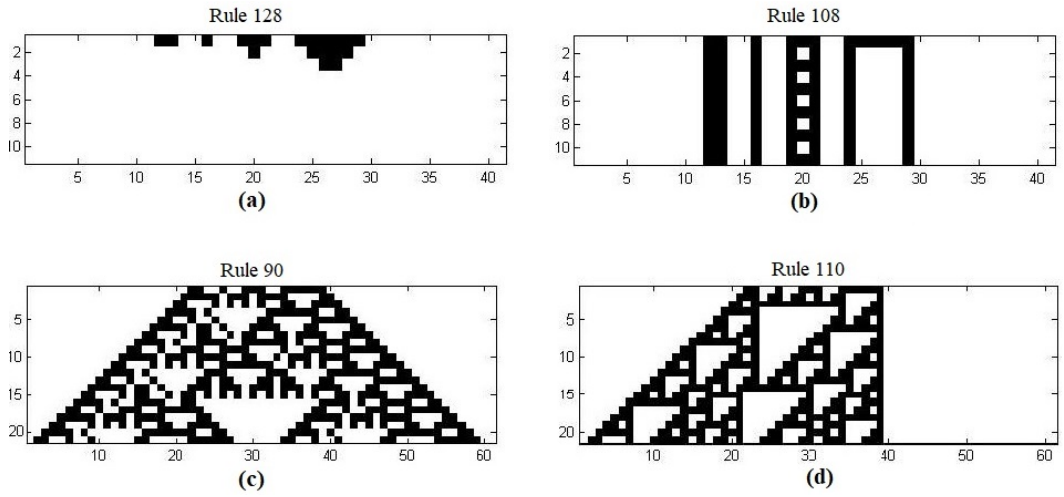


Figure 1.5 State-time diagram of some examples of ECA Wolfram Classes: (a) Class I *Uniform*, (b) Class II *Periodic*, (c) Class III *Chaotic*, and (d) Class IV *Complex* or *Edge of Chaos*.

The fourth-class rules have very high computational representation (universal computation) [74-76]. Table 1.2 illustrates the Wolfram classes of the 88 ECA equivalent sets. The most interesting rules are in the Classes III and IV because of their applications; the chaotic rules are used for random number generators and cryptography, also due to the capacity of class IV rules to contain complex systems; it has been used in the simulations of particles, waves, or gliders [36, 63, 77, 78]. In this thesis, it will be shown that the periodic rules of class II, despite their simplicity, they can also be used in machine learning, and even they give the best results in some tasks, e.g., generalized 5-bit memory task.

1.2.6 Linear Cellular Automata

1-dimensional CA is *linear*, or *additive*, if its ECA local rule (CAr) has the following form:

$$CAr(x_1, x_2, \dots, x_m) = \lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_m x_m \quad \textbf{(1.5)}$$

where the constants $\lambda_1, \lambda_2, \dots \lambda_m \in S$. If only one of the constants λ is nonzero CA is trivial, otherwise it is non-trivial [62]. For the 88 ECA different rules in Table 1.2, there are only six rules are additive as follows:

$$CAr(x_{-1}, x_0, x_1) = 0 \qquad \textit{Rule 0} \qquad \textbf{(1.6)}$$

$$CAr(x_{-1}, x_0, x_1) = x_1 \qquad \textit{Rule 170} \qquad \textbf{(1.7)}$$

$$CAr(x_{-1}, x_0, x_1) = x_0 \qquad \textit{Rule 204} \qquad \textbf{(1.8)}$$

$$CAr(x_{-1}, x_0, x_1) = x_{-1} \oplus x_0 \qquad \textit{Rule 60} \qquad \textbf{(1.9)}$$

$$CAr(x_{-1}, x_0, x_1) = x_{-1} \oplus x_1 \qquad \textit{Rule 90} \qquad \textbf{(1.10)}$$

$$CAr(x_{-1}, x_0, x_1) = x_{-1} \oplus x_0 \oplus x_1 \qquad \textit{Rule 150} \qquad (\mathbf{1.11})$$

Hence, Rules 0, 170 and 204 are trivial, and rules 60, 90 and 150 are non-trivial. Rule 0 immediately maps to zeros, rule 170 shifts the initial state one cell to the left, rule 204 is the identity, and the other rules (60, 90, 150) have complex behavior.

Mathematically, in ECA, the additive rules can be represented as linear functions modulo two, thus these rules allow to compute independently the evolution for different initial states, then the results can be combined by simply adding which significantly simplifies the hardware implementation of these rules.

1.2.7 Elementary Cellular Automata with Memory

Conventional CA are memoryless i.e., a new state of a cell depends only on the neighbors at the previous time step. A relatively new class of CA is Elementary Cellular Automata with Memory (ECAM), which is an ECA with an added memory function. This new class extended the domain of rules based on the ECA domain. In ECAM the basic ECA rule should be selected and then it is composed with a memory function. Therefore, the memory function will determine if the original ECA rule conserves the same original Wolfram's class or not. Following this principle, the ECAM rules can be classified as follows:

1- *Strong*: The memory functions are unable to change the rule class,
2- *Moderate*: The memory function can transform the rule to another class and conserve the original class as well, and
3- *Weak*: The memory function transforms the rule class to another class quickly [71, 79, 80].

1.3 Overview of Reservoir Computing based on Cellular Automata

The first appearance of reservoir computing based on Boolean networks was in 2013 using random Boolean networks RBNs [43], in 2014 [81] the RBNs have been

replaced by cellular automata CA[3] which are a special case of RBNs with a regular structure, then such framework of CA has been used in detail for connectionist machine intelligence [7] and symbolic computation [82, 83]. In previous studies [7, 81], the initial (first time step) L_{in}-dimensional input is randomly permuted R times to produce the (Lin.R)-dimensional vector X_1 (Figure 1.6) which is projected into the CA reservoir as initial state and evolved up to I iterations utilizing a certain ECA rule.

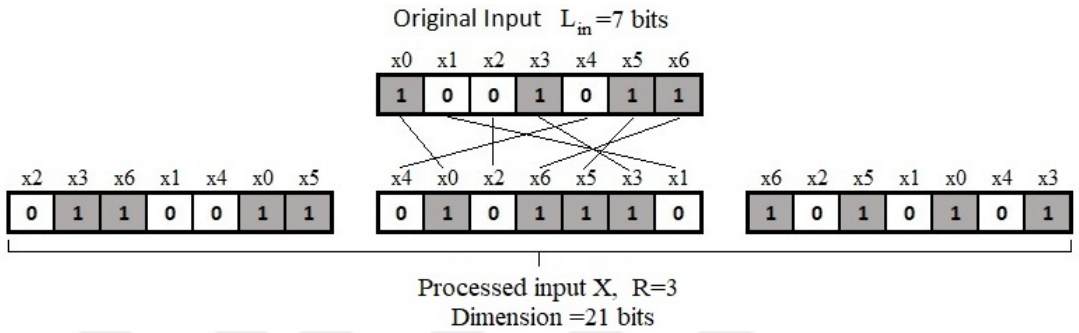


Figure 1.6 Random permutation of the original input to produce the processed input X, which is projected into the CA reservoir.

All the CA iteration states at the first time step are concatenated to produce a single feature vector[4], which is used in linear regression to find the weight values of the output weight matrix $\mathbf{W}_{out}$, that will be used to predict the output at first time step and this is analogous to ESNs in Section1.1.1. In order to insert the second time step input into the CA reservoir; the last CA iteration state of first time step is added to the next time step (L_{in}.R)-dimensional input X_2 using normalized addition[5] as an *insertion*[6] function to produce the initial state of CA reservoir at 2nd time step as shown in Figure **1.7**. Hence, the 2nd time step initial state offers the recurrent connection which memorizes the history of the first and second time step inputs. The previous procedures are repeated up to the last time step. The obtained state vectors from the CA reservoir are used to predict the output for connectionist machine learning [7, 81]. Intuitively, there should be something better than a random (input projection and normalized addition); that's why, in 2016 [8], another type of CA reservoir was designed including

[3] The physical system in Figure 1.1(b) has been substituted by cellular automata.
[4] The dimension of the feature vector is (L_{in}.R.I).
[5] In normalized addition; 0 + 0 = 0, 1 + 1 = 1 and for 1 + 0 or 0 + 1 the result is decided randomly 0 or 1.
[6] Insertion function is to insert a new input time step into the reservoir.

the use of zero array buffers instead of the input random projection in [5, 41], also replacing the normalized addition by XOR operation. This model is called ReCA (CHAPTER 2 in this thesis).

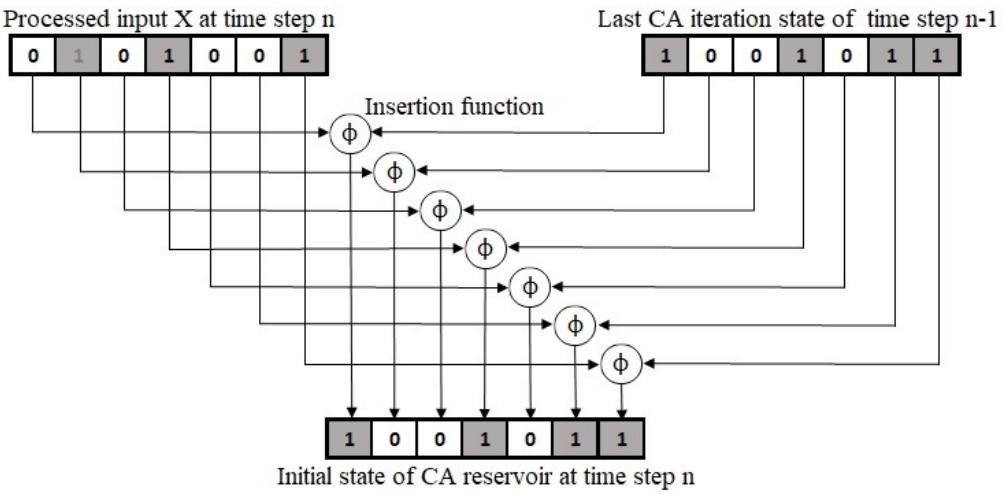


Figure 1.7 Normalized addition as an insertion function (ϕ) to create the recurrent connection which produces the next initial state of the CA reservoir.

Using zero buffers in ReCA enhances the results to solve all pathological sequence tasks, these tasks have been introduced in 1997 by Hochreiter and Schmidhuber to examine Long short-term memory (LSTM) method in [13] then they have been widely used in RNNs literature. In 2017, RC based on CA was used by applying Non-Uniform ECA (using 2 rules horizontally) [49] or by repeating the entire model in series (deep reservoir) [50]. The authors of the last two methods [49,50] have utilized the random permutation to create the initial states of CA reservoir from the original input also to insert the initial states consecutively into the reservoir. These two methods have been tested using only one type of pathological sequence tasks (5-bit memory task), which is a binary dataset. In 2017, the RC based on CA framework was tested using a nonbinary dataset by applying a multilayer CA in the reservoir; by using 2 rules vertically in [51], the first rule is for projection and the other is for memorization. In 2018, ReCA in software and hardware implementation was tested using a pattern recognition task of handwritten numbers (MNIST), which is a time-independent task [84]. In 2019 [9], new methods of ReCA feature extraction has been proposed to reduce the model complexity. The results were promising where the feature dimension

has been reduced by up to 98% in some pathological tasks (CHAPTER 3 in this thesis). Finally, in [85], the recurrent architecture with CA distributed representation of ReCA have been compared with feedforward architecture and local representation after applying new methods of feature extraction and another insertion function (CHAPTER 4 in this thesis).

1.4 Contributions

As presented in Sections 1.1.1 and 1.3; ESNs and the proposed models of using CA in RC have parameters randomly assigned and preprocessing the input by random permutation. Intuitively, there should also be something better than a *random*, this was the starting point. The contributions of this thesis are as follows:

- RC based on CA framework has been developed to produce ReCA model in which the random parameters in ESNs are replaced by an ECA rule and the input random permutation is replaced by adding two buffers to the input to enable the natural diffusion of the CA evolution states in the reservoir, finally the normalized addition (random operation see footnote 5) in Section 1.3 is replaced by XOR operation as insertion function.
- Two subroutines are added in order to preprocess the input before the reservoir:
  a- R_i expansion to reduce the interference between the nonzero elements in the reservoir.
  b- Multilayer CA to increase the nonlinearity in the model.
- Similar to ESNs the evolution states at last time step are used as feature vector for LAST method.
- Creating different options for feature extraction ALL, Each, Half, and expansion ratio *f* instead of LAST, in order to control the ReCA complexity and reduce the required training examples.
- Using the whole pathological synthetic tasks (binary dataset) to evaluate ReCA with all the above different options.
- Three different models: feedforward architecture with CA distributed representation, feedforward architecture with local representation, and recurrent architecture with local representation (stack reservoir) are provided

to compare them with ReCA (recurrent architecture with CA distributed representation) in order to find the optimum model.

- Three other options of feature extraction (XOR, Binary, Gray) are created to reduce the feature vector dimension. Also, a new insertion function (Overwrite) is proposed rather than XOR, which increases the ReCA accuracy while reducing required training examples in generalized 5-bit task, which is also created in this thesis.

To evaluate ReCA, it has also been tested using real and nonbinary tasks such as signal classification, Japanese vowels dataset [109], and IRIS dataset [118] which is time-independent. Finally, ReCA has been tested utilizing the 20 QA bAbI tasks from Facebook [120] in order to study its ability to solve these hard tasks.

1.5 Thesis Outline

The main goal of this thesis is to demonstrate that reservoir computing based on cellular automata can be used to successfully process sequence data. A suitable model called *ReCA* has been proposed and then developed using several methods to train such data and compare its performance with the state-of-the-art reservoir computing methods. Most of the original work appears in Chapters 2, 3, 4, 5, and 6.

- In CHAPTER 2, we go into the details of how ReCA can be implemented, starting by the different methods of preprocessing the input data, then we present a formalization of the CA reservoir structure, finally the read-out stage where the training parameters are estimated. The results of pathological synthetic tasks are introduced using an intuitive method of feature extraction called LAST.
- In CHAPTER 3, deferent methods of feature extraction from the reservoir are created to improve ReCA by decreasing the complexity and reducing the required training examples.
- CHAPTER 4 discusses the effects of using different architectures (feedforward or recurrent) and different data representations (local or distributed) to find the optimum model. Also, three options of feature extraction (XOR, Binary, and

Gray) are proposed to reduce the dimension of feature vector. Another insertion function (Overwrite) is also used instead of XOR in the reservoir.

- In all previous chapters, the used tasks are artificial and binary (pathological synthetic tasks), Therefore, CHAPTER 5 focuses on how can ReCA deal with real and nonbinary tasks.
- In CHAPTER 6, ReCA tackles the question answering problems (20 QA bAbI tasks from Facebook), that are very complicated and hard tasks [120].
- Finally, CHAPTER 7 gives some concluding remarks with a summary of achieved results also offers the future work.

1.6 Publications from the Thesis

- M. Margem, and O. S. Gedik, “Reservoir Computing Based on Cellular Automata (ReCA) in Sequence Learning” Journal of Cellular Automata. 14 (1-2) (2019) 153-170.

- M. Margem, and O. S. Gedik, “Feed-forward vs. Recurrent Architecture and Local vs. Cellular Automata Distributed Representation Based Reservoir Computing in Sequence Memory Learning” under review in Artificial Intelligence Review (AIRE), 2019.

CHAPTER 2

RESERVOIR COMPUTING BASED ON CELLULAR AUTOMATA (ReCA)

The setup of ReCA is presented in Section 2.1 with its three stages: encoding, CA reservoir, and read-out. In order to compare ReCA with other approaches, from the point of view of distributedness, two reservoirs with different representations of cell evolution are introduced in Section 2.2. The eight pathological synthetic tasks are defined in Section 2.3. These tasks have been used to examine ReCA by the method which is demonstrated in the experiments of Section 2.4. Section 2.5 focusses on the results of ReCA and the comparison with other approaches. Finally, the discussion is in Section 2.6.

2.1 ReCA Implementation

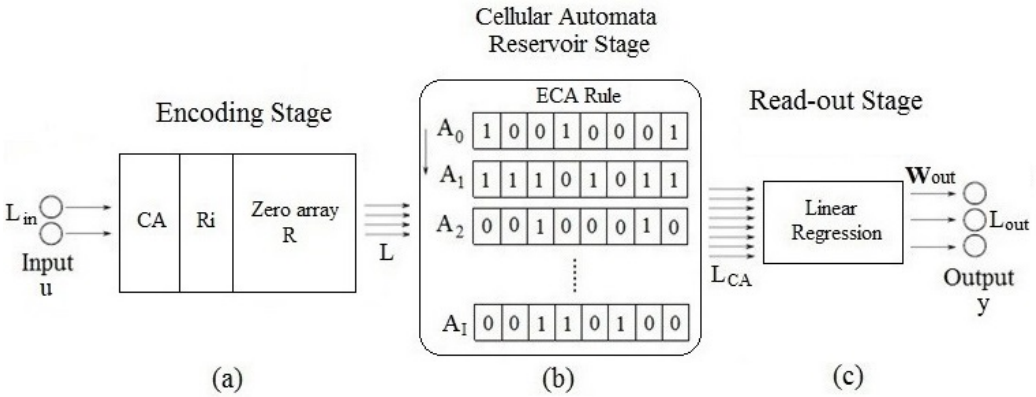


Figure 2.1 ReCA framework stages indicating vector lengths for each stage: (a) Encoding Stage composed of; (CA) Multilayer expansion using CA rules, (Ri) which expandes each input bit by Ri bits and adding (Zero Array) R-dimensional buffers for both sides of the input. (b) CA Reservoir Stage; The L-dimensional output of Encoding Stage is projected onto CA reservoir which evolves it using certain ECA rule in multiple time iterations I. (c) Read-out Stage; The reservoir output (LCA-dimensional feature vector) is trained by Linear regression to find the output weights of Wout matrix, which is used to predict the output y [8].

The ReCA framework consists of three stages as shown in Figure 2.1. The encoding stage translates the input **u** into the initial states of CA reservoir. In the reservoir stage, the ECA rules are applied for a fixed period of iterations I, in order to evolve the CA initial states. The CA states in the reservoir are concatenated to produce a feature vector that will be used in the read-out stage using Linear Regression to predict the output **y**. The details of the method are given below step by step.

2.1.1 Encoding Stage

The encoding stage can be divided into three subroutines as shown in Figure 2.1(a): CA, R_i and Zero Array R.

2.1.1.1 Utilizing Buffers (Zeros Array R)

In the encoding stage, the input **u** is translated into the initial states of a cellular automaton reservoir. For handling a sequence of inputs, each having a length L_{in}, the size of the input to the reservoir needs to be expanded. An array of zeros with length of *R* is added to both sides of original input; to hold the activity of the reservoir corresponding to previous input time steps. Then, the expanded input to cellular automata reservoir becomes of length $L=L_{in} + 2R$ as shown in Figure 2.2.

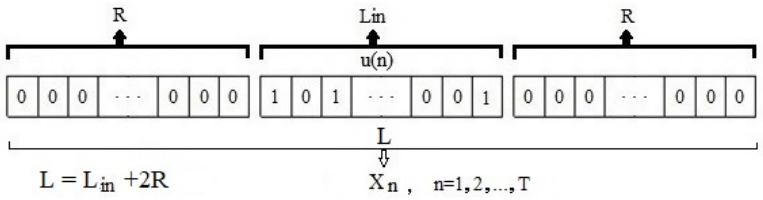


Figure 2.2 Two R-dimensional zero vectors (buffers) are added to both sides of the original input **u**(n), to produce the L-dimensional initial state X_n that will be evolved using ECA rule in the reservoir stage [85].

In most of the experiments, R=I×T (I is the number of CA iterations and T is the sequence length of the input), to guarantee that CA states have conserved all the input sequences. But, in some experiments, to reduce the time and space complexity an expansion ratio $\mathbf{f} \in [\mathbf{0}, \mathbf{1}]$ has been introduced as follows $R = f(I \times T)$, to adjust the dimension of the buffers, thus the size of the reservoir (complexity) decreases with the

value of *f*. It should be noted that only the utilizing buffers (Zero Array R) have to be used in our model, but the next two subroutines (R_i and Multilayer CA) are applied selectively according to the task requirement.

2.1.1.2 Reducing Interference R_i

To improve the accuracy in some tasks, the interference between nonzero elements in the reservoir should be reduced. The interference is due to the cancellation of cell activity when two nonzero cells collide during additive cellular automaton rules such as rule 90[7]. In order to avoid it, each bit of the input can be represented by R_i bits. At the insertion of a new time step, the location of the nonzero will rotate right one bit to reduce the interference between nonzero bits that have the *same* location in consecutive time steps as explained schematically in Figure 2.3.

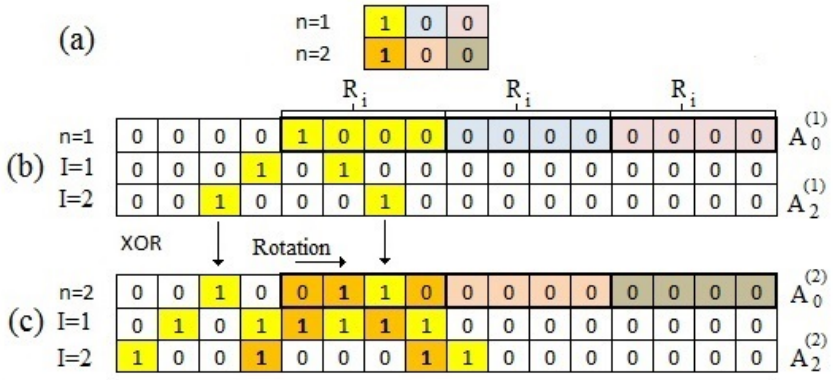


Figure 2.3 An example for how to reduce the interference between nonzero bits in rule 90 by adding R_i with rotation, when there are nonzero bits that have the same location in consecutive time steps as in; (**a**) The original 3-bit input for two time steps, (**b**) R_i = 4 bits to represent each input bit, with two iterations of CA evolution. The new input at n = 2 is represented using R_i, it might interfere with the previous input, and (**c**) But, due to the rotation, there is no interference between the 1's (in yellow) from 1st time step and the bold **1's** (in orange) from the 2nd time step.

The value of R_i should be chosen carefully; to reduce the interference between nonzero bits that have *different* locations in consecutive time steps as demonstrated in Figure

[7] Boolean expression of rule 90: $x_i(n+1) = x_{i-1}(n) \oplus x_{i+1}(n)$.

2.4. Hence, the input dimension is artificially expanded to increase the feature space size and reduce the interference between nonzero bits in the reservoir.

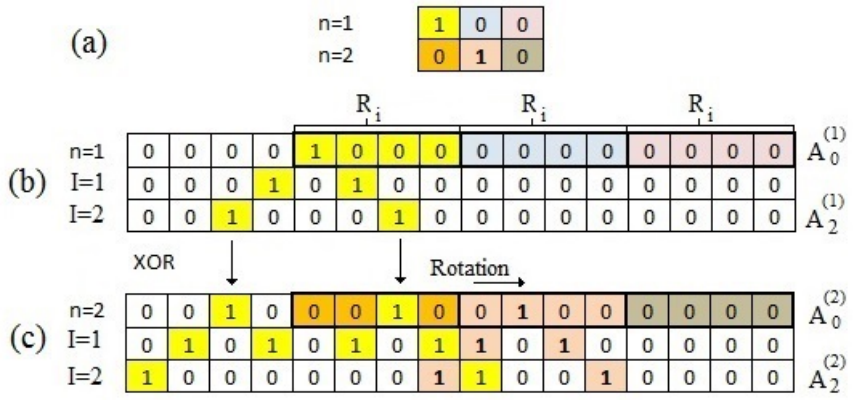


Figure 2.4 An example for how to reduce the interference between nonzero bits in rule 90, by selecting $R_i = 4$ (even value) and rotation, when there are nonzero bits that have different locations in consecutive time steps as in; (**a**) The original 3-bit input for two time steps, (**b**) $R_i = 4$ bits to represent each input bit, with two iterations of CA evolution. The new input at n = 2 is represented using Ri, it might interfere with the previous input, and (**c**) But, due to the rotation and even value of R_i, there is no interference between the 1's (in yellow) from first time step n = 1 and the bold **1's** (in pink) from second time step n = 2.

2.1.1.3 Multilayer Cellular Automata Expansion CA

Before zero paddings with right-left buffers R and expanding each binary input with R_i, the original binary input can be transformed into another binary vector using nonlinear CA rules to increase the nonlinearity of the model. Suppose that we have a binary vector of size L_{in}. We apply a nonlinear ECA rule, onto this vector for a certain amount of iterations, then use the last state of the evolution as our new input. Hence, this stage enables a *multilayer* CA architecture[8], in which the first layer projects the input into a nonlinear space, and the next layer evolves it further with linear rules in

[8] Our method should not be confused with conventional multilayer CA, they are similar only at the first-time step and they both use two different CA rules in different times (iterations). But, the multilayer CA that have been used in our work is structurally and algorithmically very different from Layered CA which is used to predict nerve axonal extension process [86] and in Cryptographic [87].

time to expand the feature space. Linearity in the second layer is essential for lossless injection of the input at each time step.

2.1.2 Cellular Automata Reservoir Stage

After the data is processed in the encoding stage as the initial states of a cellular automaton, it is passed on a CA reservoir (instead of an ESN as in [4]) for computation. The dynamics of CA provide the necessary projection of the input data onto an expressive and discriminative space, as shown in Figure 2.1(b). It was previously shown that the cellular automata reservoir holds a distributed representation of high order attribute statistics [7]. Thus, the sequence of inputs at each time step is processed to extract the input statistics and these are represented in a distributed manner as in recurrent neural networks. However, different from recurrent neural networks, for each input we can allocate a distinct memory slot in the reservoir space and avoid interference between different time steps altogether (at the cost of increased feature space). This is due to the fact that additive cellular automata rules propagate the cell activities of the previous time steps in a predictable manner, creating "empty spots" for injection of new input in the sequence.

The cellular automaton is initialized with the first time step input of the sequence (X_1) that has been obtained from the encoding stage; $A_0^{(1)} = X_1$ (with size of L), where the subscript 0 denotes initial state and increases to I, which is the number of CA iterations in the reservoir and the superscript (1) denotes to the number of time steps (from 1 to T), where T is the input sequence length. Then, the cellular automata evolution of $A_0^{(1)}$ is computed using a pre-specified ECA rule CAr up to I iterations as illustrated in Figure 2.1(b) and the following equations:

$$A_1^{(1)} = CAr(A_0^{(1)}) \quad \textbf{(2.1)}$$

$$A_2^{(1)} = CAr(A_1^{(1)}) \quad \textbf{(2.2)}$$

$$\vdots$$

$$A_I^{(1)} = CAr(A_{I-1}^{(1)}) \quad \textbf{(2.3)}$$

The evolution of CA states is concatenated to obtain a single state vector as in equation (2.4) with length of $L_{CA} = I \times L$, that will be used as a feature vector to predict the output at 1st time step:

$$A^{(1)} = \left[A_1^{(1)}\ A_2^{(1)}\ \ldots\ A_I^{(1)}\right] \quad \textbf{(2.4)}$$

At the 2nd time step, the input X_2 will be inserted into the reservoir state vector. Applying an insertion function, which is in our model, an XOR operation[9] to the last state vector $A_I^{(1)}$ with the input vector at second time step X_2. Then the new initial state vector of the cellular automaton at time step n = 2 will be as:

$$A_0^{(2)} = A_I^{(1)} \oplus X_2 \quad \textbf{(2.5)}$$

where $\oplus$ is a bitwise XOR.

Then, the cellular automaton is evolved for I steps to obtain $A^{(2)}$ as in equation (2.6).

$$A^{(2)} = \left[A_1^{(2)}\ A_2^{(2)}\ \ldots\ A_I^{(2)}\right] \quad \textbf{(2.6)}$$

$A^{(2)}$ is used for estimation at 2nd time step. This procedure is continued until the end of the sequence at n = T when X_T is inserted into the reservoir and obtaining $A^{(T)}$ that will be used to predict the output at last time step, the details are depicted in Figure **2.5**.

Note: Any feature vector $A^{(n)}$ memorizes the input information from 1st time step up to its time step n, i.e., $A^{(T)}$ memorizes the whole input information.

[9] XOR computes the correlation, which provides a lossless merging of two binary numbers: it outputs 1 if the content of input cells is different and 0 if the input cells
are identical.

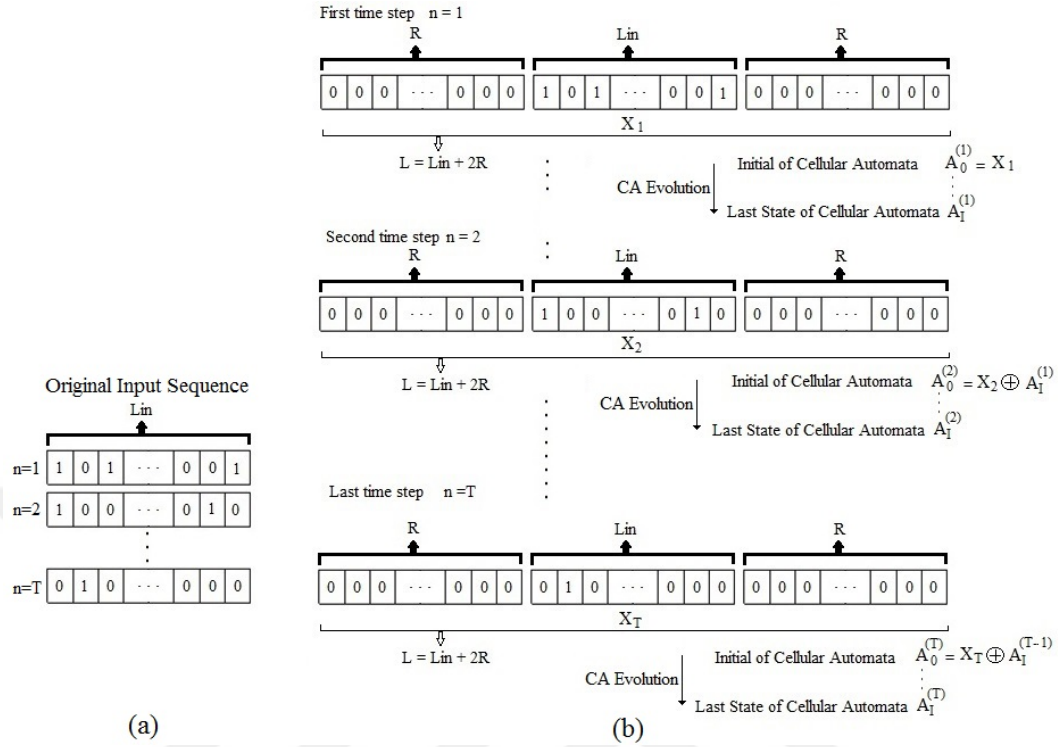


Figure 2.5 (**a**) original input sequence before the encoding stage. (b) Encoding stage and CA Reservoir stage: Adding zero array with length R to obtain the reservoir input sequence from X_1 to X_T. Then, the cellular automaton is initialized with the first time step input of the sequence, so $A_0^{(1)} = X_1$ (with size of L). The CA evolution states $A_i^{(n)}$ (i changes from 1 to I) will be used as a feature space to estimate the output y(n) using linear regression in the read-out stage [8].

2.1.3 Read-out Stage

In this stage, the output states of CA reservoir will be used as a feature vector (with dimension of L_{CA}) in the linear regression to compute the weight values of $\mathbf{W}_{out}$ matrix, which is used to predict the output. There are two cases for the output:

1- There is only *one* output **y** at last time step n = T, thus the output is a vector with dimension of L_{out}. Therefore, as in equation (1.2) for ESNs, the output **y**(T) can be expressed as follows:

$$\boldsymbol{y}(T) = \boldsymbol{W}_{out}.A^{(T)} \qquad \textbf{(2.7)}$$

where $A^{(T)}$ is the feature vector of the last time step with dimension of I.L and the size of regression parameters matrix $\mathbf{W}_{out}$ is L_{out}×IL.

2- There is an output for *each* time step, therefore the output is a matrix **Y** with size of L_{out}×T. Hence, the output **y** for each time step n > 0 is:

$$\boldsymbol{y}(n) = \boldsymbol{W_{out}}.A^{(n)} \quad \textbf{(2.8)}$$

After combining all time steps together; the feature space **A** becomes as in (2.9):

$$\boldsymbol{A} = \left[A^{(1)}A^{(2)} \ldots A^{(T)}\right] \quad \textbf{(2.9)}$$

Finally, the output will be calculated as follows:

$$\boldsymbol{Y} = \boldsymbol{W_{out}}.\boldsymbol{A} \quad \textbf{(2.10)}$$

where the size of feature space matrix **A** is IL×T and the size of $\mathbf{W}_{out}$ matrix is L_{out}×IL.

2.2 Covariance and Stack Reservoir

Other two types of reservoirs are used to study the effect of different levels of distributedness on computation in the reservoir. CA have a high distributed representation than covariance and finally stack that has local representation.

2.2.1 Covariance Reservoir

In covariance reservoir, $A^{(k)}$ is the state evolution at iteration k which is defined as in equation (2.11):

$$A^{(k)} = \prod_{-k} A_0 \oplus \prod_{+k} A_0 \quad \textbf{(2.11)}$$

where A_0 is the initial state input, $\prod_{-k}$ and $\prod_{+k}$ are permutation matrices -k and +k shifts and $\oplus$ is a bitwise XOR. $A^{(k)}$ computes the pairwise covariance of the input

attributes as in tensor products [92] and memorizes those for each sequence input. In covariance representation, there is no relation between time steps evolution with each other as in CA; there is relation only with the initial state. That is why it can be considered between CA and local representation in distributedness. The used algorithm is the same with CA algorithm, but rather than using CA rules in the reservoir, the covariance representation in equation (2.11) is used to produce the covariance evolution states, that will be utilized as a feature space to estimate the output. The covariance representation has been used in [7] on a feedforward architecture for 5-bit task, but in this dissertation, it will be applied on recurrent architecture for all pathological tasks, that will be explained in detail in Section 2.3.

2.2.2 Stack Reservoir

In stack reservoir, there is no computation involved[10]; the sequence input is memorized consecutively step by step as shown in Figure 2.6.

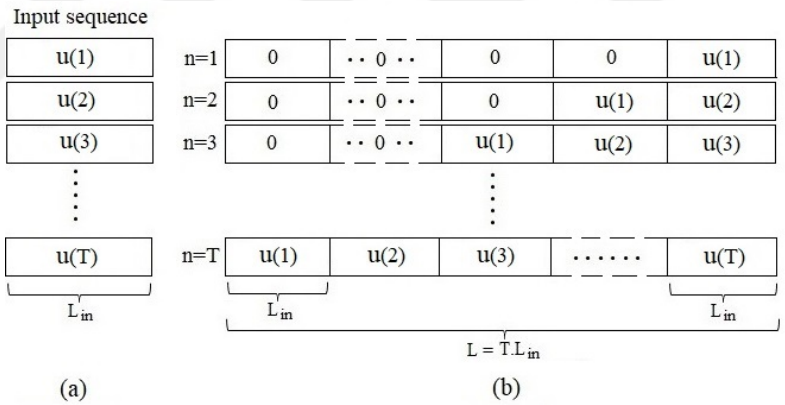


Figure 2.6 Stack reservoir (Local recurrent architecture): (**a**) The input sequence **u**, and (**b**) The input is consecutively memorized as in stack memory to produce the feature space; the first row at n = 1 is used to predict the output at first time step, the second row is used to predict the output at second time step and so on [85].

Thus, the insertion function is similar to the stack memory, the first row of feature space which has only the first time step input is used to predict the output at first time step and so on up to last time step which consists of all input time steps as demonstrated

[10] The local representation is applied only in the reservoir. Thus, the model still has a computational power from the regressor in the read-out stage.

in Figure 2.6. This representation can also be considered as a type of *variable size sliding window* that has been used in [93, 94] for frequent itemset mining and speech identification respectively, but in our case, the variation is in the amount of given information to each time-step not in the window's length.

2.3 Pathological Synthetic Tasks

The pathological synthetic problems (tasks) have been used to examine ReCA framework for long-short-term-memory capability. These tasks have been proposed by Hochreiter and Schmidhuber in [13] with minor modifications in [88], they offer long-term dependencies and are effectively impossible to solve using gradient descent [89]. They are widely used in RNNs field, e.g., [90, 91, 7, 8, 49, 50].

All these tasks are binary and can be divided into 3 categories:

1- Memory Task (5-Bit, 20-Bit and Random permutation);
2- Temporal order task (2 and 3 symbols);
3- Arithmetic/logic operation Tasks (XOR, Addition, and Multiplication).

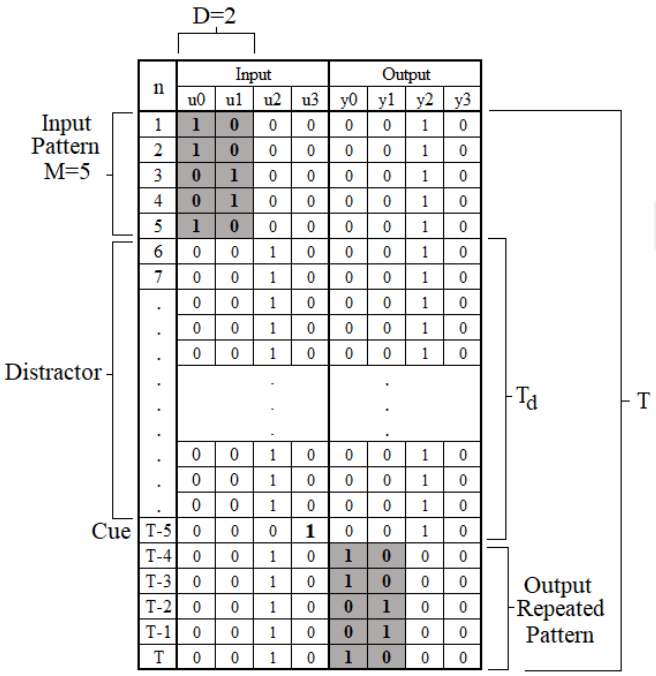
In these tasks, the input data consists of two parts: The *Information* and the *Distractor* period. The *Distractor* period is selected randomly and does not contain any information. It expands the task sequence and adds irrelevant data to increase its difficulty due to the longer-range dependencies. The output is either a particular function of the input information (XOR, Addition, Multiplication and Temporal order tasks) or repeating (memorize) the same information (from previous time steps) at final time steps (Memory and Random permutation tasks). The difficulty of these tasks increases with the value of time steps (Sequence Length) T. The following sections describe these tasks in detail.

2.3.1 Memory Tasks

Memory tasks consist of 3 types (5-Bit, 20-Bit, and Random permutation) where the input information is repeated in the output after distractor period.

2.3.1.1 5-Bit Task

In 5-bit task, the input length is 4 bits L_{in}=4, the information is a memory pattern which is the sequence of first two bits D=2 for five-time steps M=5 where one of these two bits is randomly set to 1, thus there are $2^5 = 32$ possible patterns, that's why this task is named the 5-bit task. Then, at the sixth time step, the input memory pattern is followed by a distractor [0 0 1 0] for period T_d time steps. Finally, the initial memory pattern should be repeated in the output after arriving a cue signal in the fourth input bit at the last time step of the distractor. Thus, the total sequence length of this task $T=T_d+10$. The fourth output bit is always zero, so it should be dropped, but it has been kept; because it is included in the original task. The details of 5-bit task are demonstrated in Figure 2.7.



n	Input				Output			
	u0	u1	u2	u3	y0	y1	y2	y3
1	**1**	**0**	0	0	0	0	1	0
2	**1**	**0**	0	0	0	0	1	0
3	**0**	**1**	0	0	0	0	1	0
4	**0**	**1**	0	0	0	0	1	0
5	**1**	**0**	0	0	0	0	1	0
6	0	0	1	0	0	0	1	0
7	0	0	1	0	0	0	1	0
.	0	0	1	0	0	0	1	0
.	0	0	1	0	0	0	1	0
.	0	0	1	0	0	0	1	0
.								
.	0	0	1	0	0	0	1	0
.	0	0	1	0	0	0	1	0
.	0	0	1	0	0	0	1	0
T-5	0	0	0	**1**	0	0	1	0
T-4	0	0	1	0	**1**	**0**	0	0
T-3	0	0	1	0	**1**	**0**	0	0
T-2	0	0	1	0	**0**	**1**	0	0
T-1	0	0	1	0	**0**	**1**	0	0
T	0	0	1	0	**1**	**0**	0	0

Figure 2.7 An example of 5-bit task: The input and output length $L_{in}=L_{out}=4$ bits. The first 5-time steps are the 2-dimensional input memory pattern and the last 5-time steps are the main output which is a repeated (memorized) input pattern (Shadow Bold), but we have to note that the whole output bits (4 bits) for all time steps (T) should be predicted. The distractor input [0 0 1 0] is at the middle of the task for T_d time steps, and the last time step of it is a cue signal (Bold 1 in u_3) as a mark to repeat the input pattern in the output. The total sequence length of the task is $T=T_d+10$ [85].

2.3.1.2 20-Bit Task

The 20-bit task is similar to the 5-bit; the difference is only for the size of memory pattern where D=5 bits and M=10 time steps. Hence, the input and output length $L_{in}=L_{out}=7$ bits, the total sequence length $T=T_d+20$, therefore 20-bit task is more difficult than 5-bit task. Notice that there are a set of 5 different possible patterns, which is a little bit more than 20 bits information per input sequence and has given this task its name.

2.3.1.3 Random Permutation Task.

The input in this task is 100-dimensional binary vector L_{in} = 100 bits. At each time step, one of the 100 bits gets a '1' value. At first time step n = 1, one of the first two input bits is randomly assigned to '1'. For the remaining time steps n = 2, 3, . . . , T, at each time step, one of the 98 remaining inputs is set to 1 in a random fashion. The relevant output is at the last time step at n = T; the target output is the 100-dimensional input vector at 1^{st} time step, thus L_{out} = 100 bits.

2.3.2 Temporal Order Task.

Temporal order task consists of two parts; according to the number of ordered events *two* or *three*.

2.3.2.1 2 Symbols Task.

The input vector in this task is 6-dimensional L_{in} = 6 bits, the first two inputs are for a critical event (A = [01]) or (B = [10]) and the last four inputs are distractor. There are two critical event times T_1 and T_2 associated with critical events A and B. At all-time steps except for T_1 and T_2, only one of the four distractor inputs is set to '1'. At n = T_1 the first or second input is randomly set to '1'; the same is executed for time step n = T_2.

The target output is only at the last time step, which is one of the four possible indicator outputs [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0] and [1, 0, 0, 0] according to the four possibilities of the order of A and B, thus the output length L_{out} = 4 bits. Figure 2.8 demonstrates this task in detail.

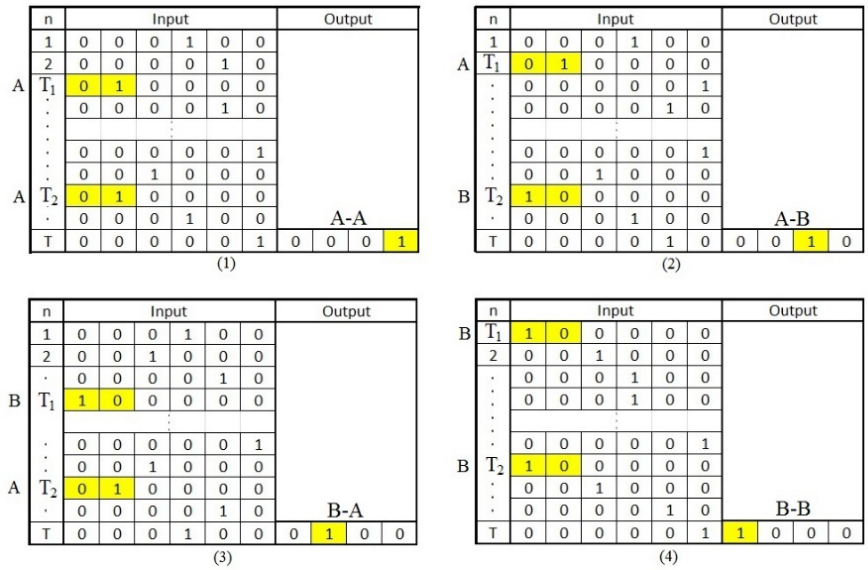


Figure 2.8 Temporal Order (2 Symbols) Task: The order of the two events A and B with the four possible indicator outputs at last time step.

2.3.2.2 3 Symbols Task.

The 3 Symbols temporal order task is completely analogous to the 2 symbols task, except that in this task there are three critical times T_1, T_2 and T_3. This makes 8 possible indicator outputs, which have to be classified by the 8 bits output at the last time step. Thus, the input dimension is still $L_{in} = 6$ bits, but the output becomes $L_{out} = 8$ bits.

2.3.3 XOR, Addition and Multiplication Tasks.

These three tasks are the same; the difference is only for the used operation XOR, addition or multiplication. As an example, the XOR task is explained in the following section.

2.3.3.1 XOR Task

In this task there are two inputs $L_{in} = 2$ bits; The first input $u_1(n)$ is a stream of zeros or ones distributed randomly. The second stream is zeros $u_2(n) = 0$ at all-time steps except at two-time steps T_1 and T_2 where $u_2(T_1) = u_2(T_2) = 1$ [11]. The objective is that at

[11] The one in second input bit is a cue signal for localization of $u_1(n)$.

the last time step where n=T (much later than T_1 or T_2); the output $y(T)=u_1(T_1)\oplus u_1(T_2)$. The task details are illustrated in Figure 2.9.

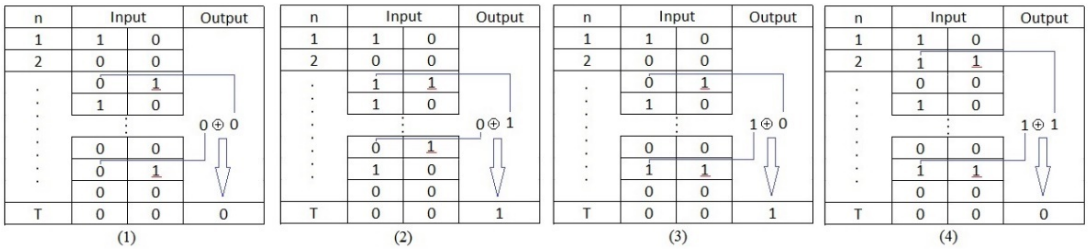


Figure 2.9 XOR Task in different situations.

2.3.4 Binary Encoded Tasks

The inputs of all pathological tasks have only one nonzero (one hot encoding) for each time step. The binary encoding is applied to the classical inputs; the location of the nonzero element will be represented by a binary number as shown in Figure 2.10, to make the task input closer to real data[12], but this leads the tasks to become harder.

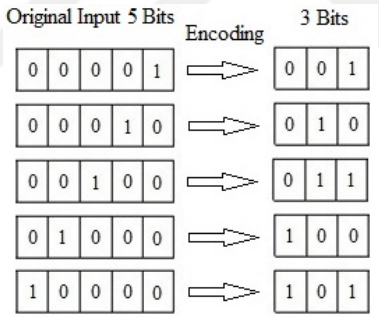


Figure 2.10 Binary encoding for payload inputs of 20-bit task, the length is reduced from 5 to 3. Thus, the total input and output length for 20-bit task are reduced from 7 to 5.

These new hard tasks have been used to test our model ReCA. The binary encoding will only be used for Random Permutation and 20-Bit tasks because the input of the other tasks is very small, so we do not need this encoding.

[12]As an example, this encoding is essential for word prediction application of language modeling, for which one hot encoded input should be of length tens of thousands (size of word dictionary).

2.4 Experiments

For all pathological tasks, a set of N_{train} (training set) and N_{test} (testing set)[13] input time sequence and its associated outputs (Target) is synthesized. ReCA is trained by the N_{train} examples to find the feature vectors $A^{(n)}$, that is used to compute the regression parameters of $\mathbf{W}_{out}$ matrix, which is used to predict the output of N_{test} examples. Then, the predicted output is compared with the original one in the testing set to evaluate the model.

2.4.1 Training Stage

After applying the input of N_{train} examples on the model and then preparing the feature vectors $A^{(n)}$ as demonstrated in Sections 2.1 and 2.2. These vectors are used to compute the linear regression weights of $\mathbf{W}_{out}$ matrix via pseudo-inverse as illustrated in equation (2.12) for the tasks that have only one output at the last time step and equation (2.13) for the tasks that have an output at each time step:

$$\boldsymbol{W_{out}} = \boldsymbol{y_{train}} \cdot (A^{(T)})^{\dagger} \qquad \textbf{(2.12)}$$

$$\boldsymbol{W_{out}} = \boldsymbol{Y_{train}} \cdot (A)^{\dagger} \qquad \textbf{(2.13)}$$

where $\mathbf{y}_{train}$ and $\mathbf{Y}_{train}$ are the output of training set (target) depend on the task, $(A^{(T)})^{\dagger}$ and $(\mathbf{A})^{\dagger}$ are the pseudo-inverse of the features that extracted from the reservoir.

2.4.2 Testing Stage

The predicted output of testing set can be obtained from equation (2.14) for the tasks that have only one output at the last time step and equation (2.15) for the tasks that have an output at each time step:

$$\hat{\boldsymbol{y}}_{\boldsymbol{test}} = \boldsymbol{W_{out}} \cdot A_{test}^{(T)} \qquad \textbf{(2.14)}$$

[13] In all experiments, $N_{test} = 100$.

$$\widehat{\boldsymbol{Y}}_{\boldsymbol{test}} = \boldsymbol{W}_{\boldsymbol{out}} \cdot A_{test} \quad (\mathbf{2.15})$$

where $A_{test}^{(T)}$ and $\mathbf{A}_{test}$ vectors are harvested from the N_{test} examples of testing set.

Note: In 5-bit task (Section 2.3.1.1), there is no testing set; due to the small value of task examples. Therefore, the whole of the 32 examples is used for training. Thus, the equation (2.15) will be used to find the predicted output of training set as in (2.16):

$$\widehat{\boldsymbol{Y}}_{\boldsymbol{train}} = \boldsymbol{W}_{\boldsymbol{out}} \cdot A \quad (\mathbf{2.16})$$

2.4.3 Model Evaluation

The dataset of the pathological tasks is binary; therefore, each bit of the predicted output $\hat{y}_i$ should be binarized as illustrated in equation (2.17):

$$\hat{y}_{bi} = \begin{cases} 0 & if \ \ \hat{y}_i < 0.5 \\ 1 & if \ \ \hat{y}_i \geq 0.5 \end{cases} \quad (\mathbf{2.17})$$

The binarized output ($\hat{y}_{(b)test}$ and $\hat{y}_{(b)train}$) will be used to find the model *errors* as follows: *testing error*$=|\hat{y}_{(b)test} - \boldsymbol{y}_{test}|$ for all pathological tasks except 5-bit task where *training error* $= |\hat{y}_{(b)train} - target|$. This *error* represents the number of false bits over all relevant time steps in output bits. The goal of our experiments is to achieve zero test error with minimal complexity, hence the parameters I (CA evolution iterations), R_i (the input expansion to reduce the interference) and f (the expansion ratio) should be tuned to their minimum values using the smallest number of training examples.

2.5 Results

This section consists of the general results, that are the comparison between the proposed three models (ReCA, Covariance and Stack), then study the effect of some parameters on the model. At the end, the comparison between ReCA and other approaches to train RNNs.

2.5.1 General Results

Table 2.1 Results for all pathological tasks using the three proposed reservoirs (CA, Covariance, Stack). The last column is the number of false bits in the predicted output of the test set. The 3rd column presents the used rule in multilayer CA with its number of iterations. The other columns are for various parameters of the proposed models [8].

Pathological Task	Method	Multilayer ECA rule[14]	T [15]	I	N_{train}	f	R_i	No of False Bits
5-Bit	ECA Rule 90	-	1000	4	32	1	1	0
20-Bit Normal	ECA Rule 90	-	300	20	120	1	2	0
	Covariance	-	300	19	120	1	3	0
	Stack	-	1000	-	5	-	-	0
20-Bit Binary Encoded	ECA Rule 90	-	200	24	250	1	8	0
	Covariance	-	100	19	250	1	2	0
	Stack	-	100	-	500	-	-	47
Random Permutation	ECA Rule 90	-	1000	2	200	1	1	0
	Covariance	-	1000	2	200	1	1	0
	Stack	-	1000	-	1200	-	-	0
Random Permutation Binary Encoded	ECA Rule 90	-	300	4	600	1	1	0
	Covariance	-	300	4	700	1	1	0
	Stack	-	300	-	1400	-	-	0
2 Symbols	ECA Rule 90	-	500	8	6000	1	1	0
	Covariance	-	200	8	6000	1	1	151
	Stack	-	200	-	6000	-	-	4
	ECA Rule 150	122 (1)	1000	8	8000	0.5	1	0
	Covariance	122 (1)	50	8	1500	1	1	56
	Stack	122 (1)	500	-	14000	-	-	0
3 Symbols	ECA Rule 90	-	50	24	7000	1	1	0
Addition	ECA rule 150	40 (1)	500	20	2500	0.5	1	0
	Covariance	110 (1)	50	20	3300	1	1	6
	Stack	110 (2)	1000	-	3500	-	-	0
Multiplication	ECA rule 150	110 (2)	500	12	2000	0.5	1	0
	Covariance	110 (2)	50	12	2000	1	1	17
	Stack	110 (2)	1000	-	14000	-	-	0
XOR	ECA rule 150	40 (1)	1000	4	500	0.5	1	0
	Covariance	110 (2)	50	4	3200	1	1	35
	Stack	110 (2)	50	-	3200	-	-	39

[14] The number between brackets is the number of iterations that are used in multilayer CA.
[15] T is the input sequence length except for 5-bit and 20-bit tasks where it means the distractor length T_d.

Table 2.1 collects the results of all pathological tasks using the three proposed models (CA, Covariance, and Stack). ReCA has solved all pathological tasks **a**. *directly*: in random permutation, 5-bit and temporal order tasks, or **b**. by *expanding* the input either utilizing R_i to reduce the interference in 20-bit task or using the CA multilayer in addition, multiplication and XOR tasks.

Covariance and Stack representations could solve only memory and random permutation tasks, but Stack fails in binary encoded 20-bit task. Covariance could not solve the temporal order tasks and arithmetic/logic operator tasks while Stack could solve them using multilayer CA expansion except XOR. Surprisingly, the Stack reservoir achieves zero test error up to 1000 time steps with $N_{train} = 5$ for 20-bit task, which is a very hard task in [88, 90, 91], even though the stack model is very simple. The superior results that are seen for ReCA are due to its large distributed representation and higher attribute statistics.

2.5.2 The Effect of Training Examples N_{train}

Providing the model more information, by increasing the number of training examples N_{train}, improves the accuracy as illustrated in Table 2.2 for random permutation and 20-bit tasks using rule 90.

Table 2.2 The increasing of N_{train} improves the accuracy in ReCA: where, $N_{test} = 100$ and $f = 1$, for 20-bit task ($T_d = 300$, I = 20 and Ri = 2), and random permutation task (T = 1000 and I = 2). Using rule 90.

20-Bit Task		Random Permutation Task	
Ntrain	No of False bits	Ntrain	No of False bits
50	4	100	22
100	2	150	2
120	0	200	0

2.5.3 The Effect of Sequence Length T

The increasing of sequence length T increases the range of temporal dependencies, that makes the task harder. Therefore, the task then requires more training examples to obtain zero test error as experimentally proved in Table 2.3 for random permutation binary encoded task as an example using rule 90.

Table 2.3 The minimum training examples to achieve zero test error w.r.t. the increasing of sequence length T. For random permutation binary encoded task, where I=4 and *f*=1 using rule 90.

Sequence Length T	min N_{train}
30	100
50	150
100	200
200	300
300	600

2.5.4 The Effect of the Expansion Ratio *f*

The increasing of expansion ratio *f* reduces the number of false bits due to the increasing of the feature vector dimension, i.e., increasing the provided information to the regressor. As illustrated in Table 2.4, the zero error is obtained at $f = 0.3$ in the 5-bit task, this means; the feature space can be reduced to 0.3 of its maximum value without losing the model accuracy.

Table 2.4 The effect of expansion ratio on the accuracy (No of False bits) in 5-bit task.

Expansion ratio *f*	No of False bits
0	224
0.05	58
0.1	32
0.15	32
0.2	32
0.25	8
0.3	0
0.35	0
⋮	⋮
1	0

2.5.5 Multilayer CA Expansion

Table 2.5 indicates that for XOR task, all the *nonlinear* ECA rules from class IV with a complex behavior could achieve zero test error, thus they can be used for input multilayer expansion with some of rules from other classes; to increase the nonlinearity in the model. But, on the contrary, all *linear* (additive) ECA rules could not solve the XOR task. This result is predicted; because the reservoir utilizes a linear rule, hence

any addition of linear rule in multilayer expansion is the same with increasing the iterations in the reservoir using the original rule, which already cannot solve the task by itself.

Table 2.5 The No of False bits in XOR task; using various ECA classes and rules in the multilayer expansion.

ECA Class	Linear Rules		Nonlinear Rules	
	ECA Rule	No of False bits	ECA Rule	No of False bits
I	250	73	40	0
			32	45
II	170	73	118	0
	204	77	138	76
III	60	71	122	0
	90	66	45	62
	150	80	×	×
IV	×	×	41, 54, 106, 110	0

2.5.6 One Hot Encoding

Instead of using multilayer CA expansion, it can be used other type of input transformation such as one hot encoding, which significantly improves the results of XOR task. In one hot encoding, the original task input $L_{in} = 2$ will be expanded to $L_{in} = 4$; as follows [00] → [0001], [01] → [0010], [10] → [0100] and [11] → [1000].

ReCA alone could not solve the original XOR task, but after using the one hot encoding, the task has been easily solved using rule 150 with small effort (I=2 and N_{train}=50) and even without using the buffers R in the encoding stage, i.e., the expansion ratio f=0.

2.5.7 Comparison with Other Approaches

The ReCA results outperform: **1**. [88] (2011) and [91] (2013) where the zero test error has been obtained for sequence length T ranging from 50 to 200 time steps, while in our experiments T ranging from 200 to 1000 time steps. **2**. [90] (2012) where the zero test error could not be achieved in 20-Bit binary encoded task using ESNs even if a large reservoir has been used and after changing the weights of input matrix as listed in Table 2.6. Also, ReCA outperforms ESN in computational complexity for most of the tasks as illustrated in

Table 2.7. Comparing the representations of ReCA and ESNs where ReCA outperform in 20-bit and 3 Symbols tasks but ESNs outperform in 5-bit task. Also, the computation performed in ReCA is much more transparent for analysis and improvement, while in ESNs the state evolution is untraceable due to random and irregular distributivity.

Table 2.6 20-Bit task with binary encoded input using ESNs: The No of false bits for different values of reservoir size K at T_d = 200, Ntrain = 500 and Ntest = 100. For the input matrix weights in Basic ESNs $\sigma_{1,2,3} = 2.5 \times 10^{-6}, \sigma_4 = 1 \times 10^{-6}$, and $\sigma_5 = 1$. For more details about Blind, Basic conditions, reservoir size (K) and input matrix weights (σ) in ESNs please refer to [90].

Reservoir Size N (neurons)	No of False Bits	
	Blind	Basic
2000	50	27
4000	46	15
6000	27	24
8000	25	66
10000	25	59
15000	81	55

Table 2.7 The comparison of the number of operations and correspondence required bits between ESNs and ReCA for examples of pathological tasks [8].

Task	ESNs (Floating point ⇨Bits)	ReCA	Speedup
20-Bit, Td=200	105.6M ⇨ 3380 Mbit	24.8 Mbit	136X
3 Symbols, T=200	5M ⇨160 Mbit	20.5 Mbit	7.8X
XOR, T=1000	0.2 M ⇨ 6.4 Mbit	16 Mbit	0.4X

2.6 Discussions

The proposed framework (ReCA) could solve all the pathological tasks either *directly* or after *expanding* (preprocessing) the input using multilayer CA and/or R_i to reduce the interference between the nonzero elements in the reservoir.

ReCA outperforms the Covariance and Stack representations because it has the highest distributed representation, while Stack representation provides only pure memorization, and Covariance representation computes only second order statistics.

Usage of cellular automaton instead of real-valued neurons in ESNs greatly simplifies the architecture and makes the computation faster and more transparent for analysis.

The linear (additive) *ECA* rules (90 and 150) have been used in the reservoir evolution, to achieve lossless injection of input at each time step, because the linearity of these rules maximizes one-to-one correspondence between the input sequence and the reservoir activity due to the sequence. Moreover, the additive rules can be represented as linear functions modulo two, thus these rules allow to compute independently the evolution for different initial states, then the results can be combined by simply adding which significantly simplifies the hardware implementation of these rules.

On the other hand, *ReCA* outperforms conventional methods of reservoir computing including *ESN*s by:

1- Using binary numbers instead of floating point numbers,

2- Multiplication in *ESN*s being replaced with bit-wise logic in the *ReCA* reservoir and multiplication in the regressor being replaced with summation due to the binary data,

3- The C*A* reservoir hardware can be implemented using ordinary digital gates or field-programmable gate arrays FPGAs. Hence, the model complexity (space, time) and the power consumption are reduced.

CHAPTER 3

COMPLEXITY REDUCTION OF ReCA

The model selection should be based not only on the accuracy of fit, but the model complexity must also be taken into account. A simple model will generalize better in new data sets than a complex one and thus will have more predictive accuracy. In addition, a simple model's behavior is more tractable because parameter estimates will be more stable. Therefore, it is necessary to consider the complexity in model evaluating [95]. Finally, the complex models are very expensive in space and time for a program running. That is why, in this chapter, we will deal with ReCA complexity by creating new methods to extract the features from CA reservoir. Then, observing the effect of using these methods on the ReCA model complexity.

Most parts of this chapter are published in [9].

3.1 ReCA Implementation

The ReCA framework (Figure 2.1) in Section 2.1 is also used in this chapter with its three stages: Encoding, CA reservoir, and Read-out. The contributions in this chapter are the new methods that are created to extract the features from the reservoir to be used in read-out stage in order to compute the regression weight of $\mathbf{W}_{out}$ matrix, which is used to predict the output.

3.1.1 Feature Extraction from the Reservoir

In CHAPTER 2, the reservoir states of the last time step have only been utilized as a feature space, which memorizes all input information at the last time step, after which it is used to predict the output at the same time step. In this chapter, ReCA will be improved by creating new methods to extract the features from the reservoir while maintaining the high performance of ReCA. This improvement is performed in two ways:

1- The *first* is by reducing the CA reservoir size, thereby leading also to a decrease in the input size of the read-out (linear regression) stage.

2- The *second* way is by using all time steps to predict the last time step output, i.e., increasing the feature space to reduce the training examples that are required for a zero test error.

The first way can be introduced into the second one to obtain a distinguish results as depicted in the results Section 3.3.

The feature expansion can also be categorized into two types: essential and supplementary.

3.1.1.1 Essential Feature Extraction

In order to predict the output at last time step $\mathbf{y}(T)$; the model can use the CA evolution states (iterations) in the reservoir at the last time step as a feature space. Then, these states are concatenated to produce a single feature vector $A^{(T)}$ (with dimension of $L_{CA}=I.L$). This method has been used in the previous chapter, from now, it will be called LAST because ReCA uses the CA evolution states only at the *last* time step.

The feature space can be increased by choosing the CA evolution states at all time steps from n=1 to n=T. Then, all of these states are concatenated to produce a large single feature vector $A^{(ALL)}$ (with dimension of $L_{CA} = I \times L \times T$). This method will be called ALL and used to predict the output at last time step $\mathbf{y}(T)$. ALL method can be considered as an extra memory resource in RNNs as in [96-99].

We should not confuse the feature *vector* $A^{(ALL)}$ with the feature *matrix* $\mathbf{A}$ which has a dimension of $L_{CA}=I.L\times T$ and has been used in Section 2.1.3 in equation (2.10) to predict the output matrix $\mathbf{Y}$. Hence, this method is LAST not ALL because the features that have been used to predict the output $\mathbf{y}(n)$, at time step n, have only been extracted from the CA evolution states at a single time step n (not from all time steps as in ALL method). For more details please refer to Section 2.1.3.

ALL and LAST methods are *essential* because ReCA should use one of them to extract the features from the reservoir.

3.1.1.2 Supplementary Feature Extraction

The CA evolution states in the reservoir at a certain time step n can be expressed as a matrix ($\mathbf{CA}_{out}$) with I rows, and L columns as exhibited in Figure 3.1. The size of the feature space that obtained from ALL or LAST methods can be reduced; using *three* options:

1- Each: Selecting only the last k rows out of all the I rows to be used as a feature space as described in Figure 3.1. Thus, the new dimension of the feature vector becomes as follows:

$$L_{CA} = L.k = (L_{in} + 2R).k \quad \textbf{(3.1)}$$

where L_{in} is the dimension of the original input (before encoding stage) and R is the dimension of the added buffers in the encoding stage.

2- Half: The dimension of the feature space can be reduced to half by using only the right or left side of $\mathbf{CA}_{out}$ columns as in Figure 3.1. Thus, the feature vector dimension is reduced by using only one buffer R as shown in equation (3.2)

$$L_{CA} = L.I = (L_{in} + R).I \quad \textbf{(3.2)}$$

3- Expansion ratio *f*: The feature space can also be reduced by selecting only the middle columns of CA_{out} matrix (Figure 3.1), i.e., reducing the dimension of the two buffers. Hence, the buffer dimension $R = f(I \times T)$ where the expansion ratio $f \in [0,1]$.

Notes:

1- The options (Each, Half and *f*) should be used with one from the two methods (LAST and ALL).

2- Due to the rich dynamics that are provided by the CA reservoir; the three options (Each, Half and *f*) can be used together to obtain a significant complexity reduction as will be shown later in the results.

3- The (Each, Half and *f*) options are supplementary feature extraction. Hence, it is possible to confine only LAST or ALL methods, but with large feature space.

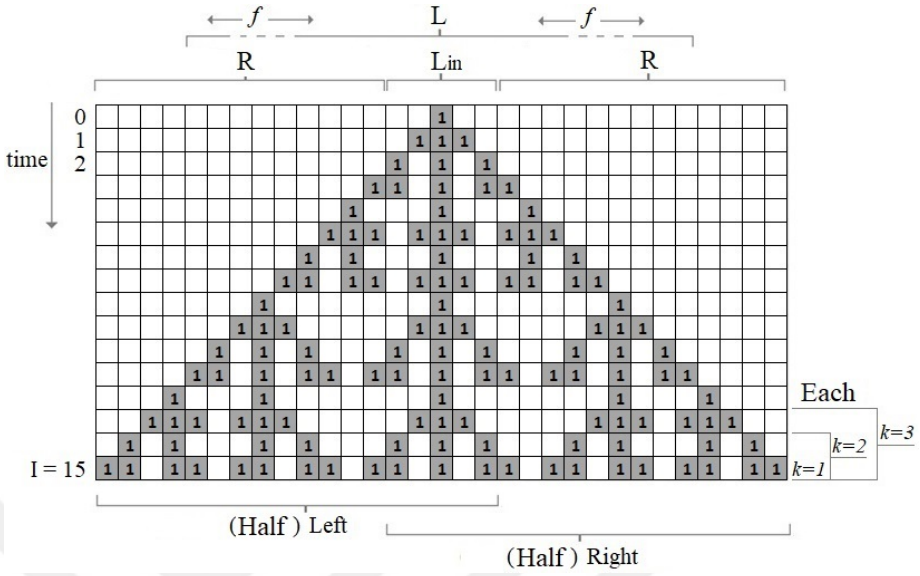


Figure 3.1 Feature extraction from the space-time diagram of the CA evolution states in the reservoir at a certain time step n (CA_{out} matrix). As an example, ECA rule 150 is used with a single `1' initial state and 15 iterations. Only k iterations can be used as features in Each option and/or using only the Half (Right or left) side of CA_{out} matrix as features, and/or using the features after reducing the columns of CA_{out} matrix by selecting expansion ratio $f < 1$ [9]. Note: White squares represent the zeros.

3.2 Experiments

All the pathological synthetic tasks in Section 2.3 have been used to examine ReCA utilizing the new feature extraction methods in the previous section. Following the same procedures of the experiments[16] in Section 2.4; to evaluate ReCA from the point of view of complexity using the new methods of feature extraction. In our experiments, the feature vector dimension L_{CA} indicates the complexity, which is used to evaluate the model. Then, comparing our new results with the classical results in Section 2.5.1.

[16] The difference is only how to select the feature vector $A^{(n)}$, which is explained in detail in Section 3.1.1.

3.3 Results

In order to compare between the feature extraction methods in ReCA for space and time complexity, the feature vector dimension L_{CA} will be used as a measure to this complexity as in Table 3.1. But, for the training stage, L_{CA} is not enough because the linear regression which is used in the read-out stage depends on the number of training examples N_{train}. Thus, the size of Linear Regression inputs (LRi) is also important, where $LRi = L_{CA} \times N_{train}$, hence LRi will be used as an indicator of complexity in the training stage. However, in the testing stage, only the L_{CA} is important because of a fixed value of testing examples $N_{test} = 100$ has been used in all our work for a fair comparison.

Table 3.1 Feature vector dimension L_{CA} for several methods of feature extraction from the CA reservoir where L_{in} is the length of the original input (before the encoding stage), T is the input sequence length, R_i is the input expansion to minimize the interference, R is the buffer dimension, I is the iterations in the CA reservoir, k is the selected states in the **Each** option, L is the length of the CA reservoir input (after the encoding stage) and *f* is the expansion ratio. Notes: **a**- The first column (LAST) is the classical method that has been used in the previous chapter [8] and it will be compared with the other new methods that proposed in this chapter [9]. **b**- For L_{CA} of the Half option, it is dependent on the condition that will be used LAST, ALL or Each [9].

Parameters	Method				
	LAST [17]	LAST-Each	ALL	ALL-Each	Half
L_{CA}	$L \times I$	$L \times k$	$L \times I \times T$	$L \times k \times T$	
L	$(L_{in} \times R_i) + 2R$				$(L_{in} \times R_i) + R$
R	$f \times I \times T$				

In our experiments, we are seeking minimum complexity, i.e., the minimum value of (iterations I, selecting states in Each option k, and the expansion ratio *f*) that will produce a minimum feature vector dimension L_{CA} in bits to achieve zero test error using minimum training examples N_{train}. The expression "the model could solve the task" means that the model could predict correctly (with zero test error) all the output bits in all time step[18] for the whole 100 examples of the testing set.

[17] In 5-bit and 20-bit tasks with LAST, $L_{CA} = L \times I \times T$ because there is an output at every time step.
[18] It is dependent on the task; if it has an output at every time step as in 5-bit and 20-bit tasks, or there is an output at only last time step (the other tasks).

3.3.1 5-Bit and 20-Bit Tasks

In 5-bit task, the zero train error can be obtained with up to a 94 percent reduction of LRi using three states k = 3 out of I = 4 with a small expansion ratio f = 0.15 and only one side of the CA evolution states (Half option) as listed in Table 3.2.

Table 3.2 5 and 20-Bit Tasks: The reservoir parameters of the best results to achieve zero test error using different methods for feature extraction. L_{CA} and LRi are in bits, %Test and %Train Reducing are the reducing percentages of L_{CA} and LRi between the new methods in this chapter (the methods in [9]). and the LAST method in CHAPTER 2 (the method in [8]). The other parameters are explained in Table **3.1**. The first column presents the results of the LAST method used in [8] and we compare its results with other new methods. The bold values are the best obtained results (minimum dimension for L_{CA} and LRi) and the minimum number of training examples N_{train} to achieve the zero test error [9].

Parametres	5-Bit Task			20-Bit Task		
	Rule 150, T=200, Lin=4, I=4, and R_i=1			Rule 90, T=50, Lin=7, I=16, and R_i=8		
	LAST	LAST-Half	LAST-Each-Half	LAST	LAST- Each	LAST-Each-Half
f	1	0.15	0.15	1	0.9	0.9
k	4	4	3	16	1	3
N_{train}	32	32	32	150	**100**	300
L_{CA} (bits)	1.28M	99.2K	74.4K	1.33M	74.8K	116K
%Test Reducing	-	-	-	0	**94**	91
LRi (bits)	41.1M	3.17M	2.38M	200M	7.48M	35M
%Train Reducing	0	92	94	0	**96**	82

For 20 bit task, L_{CA} and LRi can be reduced to 94% and 96%, respectively, using only k = 1 states out of I = 16 with an expansion ratio of f = 0.9. Moreover, the required training examples to achieve zero test error decreased from 150 examples in LAST to 100 examples in LAST-Each. However, if the Half option is used, the required training examples increase to 300 examples and k increases to 3; due to the decreasing of features to the half as listed in Table 3.2.

Figure 3.2 shows that, for the 20-bit task, LAST method requires minimum training examples to attain zero test error due to the large amount of information that it has, compared with LAST-Each-Half where using only 3 states out of 16 with half space. Using the right or left side is almost the same in this task.

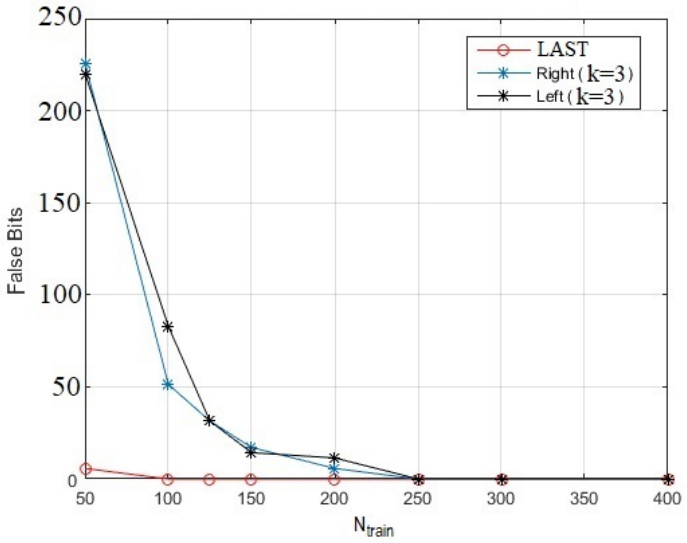


Figure 3.2 20 Bits Task: No of false bits vs N_{train} for LAST and LAST-Each-Half (Right and Left) methods, where k = 3, I = 16, T = 50 and Ri = 8, using Rule 90.

3.3.2 Random Permutation Task

The best results are reached using different options, such as a 74 percent reduction in L_{CA} using the LAST-Each-Half method and a 48 percent reduction in the LRi using the LAST- Half case, whereas the minimum training examples was 60 from the ALL-Each option, as illustrated in Table 3.3. The zero buffers R can be removed in the ALL case, but the removing of the most state evolutions in the CA reservoir leads to an increase in the required training examples to 150 to solve the task. Moreover, in the minimum L_{CA} case (LAST-Each-Half), the maximum value of N_{train} = 650 is required to obtain the zero test error to overcome the under-fitting caused by a small value of features, as shown in Table 3.3.

Figure 3.3 also presents that the zero buffers R can be removed (f = 0) in ALL method and achieve zero test error, but for LAST option to obtain zero test error, the zero vector buffers R should be kept complete, i.e., $R = I \times T$ to solve this task.

The increase in the sequence length T makes the task harder due to the longer time dependencies; that is why ReCA needs more training examples to solve the task as illustrated in Table 3.4. The trend of increasing N_{train} is not the same between ALL and

LAST methods where in LAST the requiring of training examples is large enough to increase the cost of programming running, even the laptop exceeds its RAM limitation.

Table 3.3 Random Permutation Task: The reservoir parameters of the best results to obtain the zero test error using different methods for feature extraction. The parameters are explained in Table 3.1. If the %Test or %Train Reducing is greater than 100%, it means our result is greater than the result in [8]; i.e., there is no reduction. The bold values are the best obtained results [9].

Parametres	Rule 90, T=500, L_{in}=100, I=2, and R_i=1				
	LAST	ALL	ALL-Each	LAST-Half	LAST-Each-Half
f	1	0	0.85	1	1
k	2	2	1	2	1
N_{train}	90	150	**60**	90	650
L_{CA} (bits)	4.2K	100K	900K	2.2K	1.1K
%Test Reducing	0	2381	21429	48	**74**
LRi (bits)	378K	15M	540M	198K	715K
%Train Reducing	0	3968	14286	**48**	189

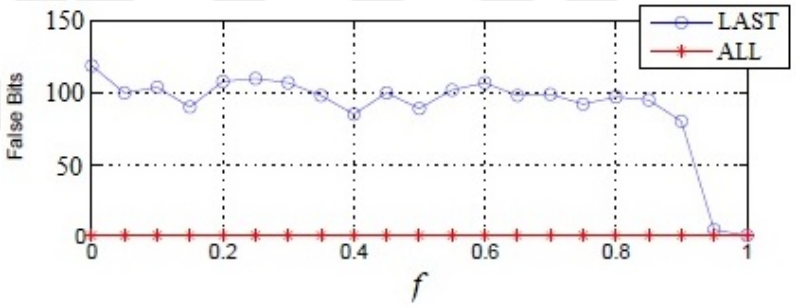


Figure 3.3 Random Permutation Task: No of False Bits vs the expansion ratio *f* using rule 90 for **ALL** and **LAST** methods. Where, I = 2, T = 100, N_{train} = 100 and N_{test} = 100.

Table 3.4 Random Permutation Task (Binary Encoded): The effect of sequence length increasing T on the minimum number of training examples to attain zero test error, using LAST and ALL methods.

T	Minimum N_{train}	
	LAST	ALL
30	100	35
50	150	40
100	200	100
200	300	100
300	600	100
500	>2000 (Out of Memory)	150

Based on Table 3.5, the minimum feature vector dimension is obtained using the LAST method with 90 examples for training, to reduce the training examples, ALL or

All-Each can be used but with increasing of the feature vector dimension. Hence, there is a trade-off between the number of training examples and feature space size. The worst case is ALL with zero expansion ratio *f*=0 because it can be replaced by LAST with better performance.

Table 3.5 Random Permutation task: No of False Bits and Feature vector dimension L_{CA} w.r.t. N_{train} for ALL, LAST and ALL-Each methods with different values of expansion ratio *f*, where T=500, I=2 and N_{test} = 100 using rule 90.

Method	*f*	N_{train}	No of False bits	L_{CA} (bits)	LRi (bits)
ALL	0	130	12		
		140	1		
		150	0	100K	15M
ALL	1	40	55		
		50	6		
		60	0	2.1M	126M
ALL-Each k=1	1	40	16		
		50	2		
		60	0	1.05M	63M
LAST	1	70	11		
		80	4		
		90	0	4.2K	378K

3.3.3 Temporal Order Tasks

Table 3.6 shows that the best results are obtained using ALL-Each-Half except that the minimum L_{CA} for the 2 symbols task was in the LAST-Half case.

Table 3.6 Temporal Order Tasks: The reservoir parameters of the best results to obtain the zero test error using different methods for feature extraction, where T = 50, L_{in} = 6 and R_i = 1. The parameters are explained in Table 3.1. If the %Test or %Train Reducing is greater than 100%, it means our result is greater than the result in [8]; i.e., there is no reduction. The best results are in bold [9].

Parametres	2 Symbols Task			3 Symbols Task	
	Rule 150, I=8			Rule 90, I=24	Rule 150, I=8
	LAST	LAST-Half	ALL-Each-Half	LAST	ALL-Each-Half
f	1	0.75	0.5	1	0.85
k	8	8	1	24	1
N_{train}	900	800	**110**	7000	**500**
L_{CA} (bits)	6.45K	2.45K	10.3K	57.7K	17.3K
%Test Reducing	0	**62**	160	0	**70**
LRi (bits)	5.8M	1.96M	1.13M	400M	8.65M
%Train Reducing	0	66	**80**	0	**98**

In 2 symbols task, ALL-Each method outperforms ALL and LAST where zero test error has been achieved using minimum expansion ratio *f*=0.15 while *f*=0.35 for ALL and LAST as depicted in Figure 3.4.

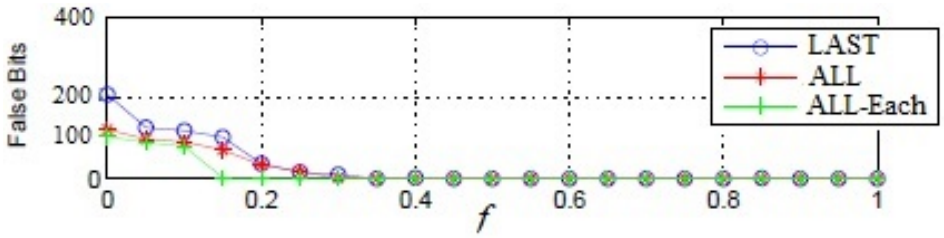


Figure 3.4 2 Symbols Order Task: No of False Bits vs expansion ratio *f* where T = 50, I = 16 and N_{train} = 900 using rule 150.

Figure 3.5 shows that ALL and LAST methods could not solve the 3 symbols task using small training examples $N_{train} < 800$ examples while ALL-Each and ALL-Each-Half could solve such task starting by $N_{train} = 500$ examples. ALL-Each and ALL-Each-Half are almost the same, therefore it is preferred to use ALL-Each-Half due to its small feature vector dimension.

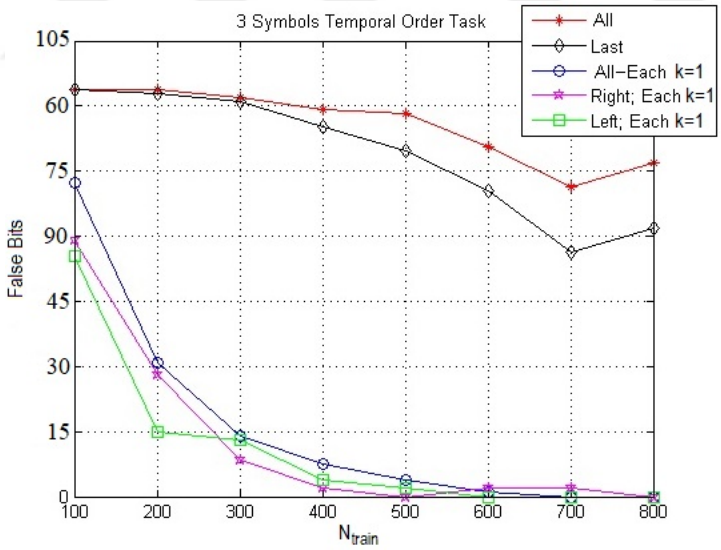


Figure 3.5 3 Symbols Temporal Order Task: No of false bits vs N_{train} for ALL, LAST, ALL-Each and ALL-Each-half (Right, Left) methods, where T = 50, I = 8 and k = 1 using rule 150.

In temporal order tasks, the LAST method has the smaller feature vector dimension L_{CA}, but with large value of examples for training. On the contrary, ALL-Each method has the larger feature vector dimension L_{CA}, but with small number of examples for

training which leads to small LRi as illustrated in Table 3.7. Thus, to choose the best method, it depends on which is the priority; the feature vector dimension or the number of training examples and the complexity of the training process.

Table 3.7 Temporal Order Tasks: different parameters for ALL-Each and LAST methods to achieve zero test error using rule 150.

Task	Method	N_{train}	T	I	f	L_{CA} (bits)	LRi (bits)
2 Symbols	ALL-Each	800	1000	16	0.25		
	ALL-Each	350	200	8	0.35	225K	78.8M
	LAST	3500	200	8	1	25.6K	89.8M
3 Symbols	ALL-Each	2000	200	16	1		
	ALL-Each	700	50	8	1	40.3K	28.2M
	LAST	7000	50	24	1	57.7 K	404 M

3.3.4 XOR Task

The multilayer CA expansion should be used to solve this task. Rule 40 has been applied to the original input with Rule 150 to the reservoir. The best results for L_{CA} and LRi are in the LAST-Half option, but the minimum required training examples was obtained using the ALL-Each-Half case, as shown in Table 3.8.

Table 3.8 XOR Task: The reservoir parameters of the best results to obtain the zero test error using different methods for feature extraction, where the Multilayer CA Rule is 40 and the number of iterations in the Multilayer CA stage is I-Multilayer = 1. The parameters are explained in Table 3.1. If the %Test or %Train Reducing is greater than 100%, it means our result is greater than the result in [8]; i.e., there is no reduction [9].

Parametres	Rule 150, T=50, L_{in}=2, I=4, and R_i=1		
	LAST	LAST-Half	ALL-Each-Half
f	0.5	0.5	0.5
k	4	4	1
N_{train}	100	60	**50**
L_{CA} (bits)	816	416	5.2K
%Test Reducing	0	**49**	637
LRi (bits)	81.6K	25K	260K
%Train Reducing	0	**69**	319

3.4 Discussions

In Chapter 2 [8], ReCA used only the *last* time step of the CA evolution states in the reservoir for all iterations I to train RNNs (LAST method). But, in this chapter other methods for training have been introduced to improve ReCA as listed below [9]:

1. All CA evolution states are used for all time steps (ALL method) in which the learning requires a small training set, but with large feature space and possibility of overfitting due to the increasing of model complexity.

2. The method ALL-Each is similar to ALL but the last k iteration states are only used from all CA evolution iterations I for all time steps. Thus, the feature space dimension decreases which leads to a reduction in the model complexity and then reduction in the possibility of overfitting.

3. The dimension of zero vector buffer R is reduced by selecting small values of expansion ratio *f*. This reduction is only suitable for ALL and ALL-Each methods, because of their large size, but in LAST, it reduces the model efficiency; to overcome such issue, a very large training set is required.

4. The *right* or *left* side of CA evolution states are only used for training in the three methods (LAST, ALL, and ALL-Each). The results show symmetry in feature space; therefore, the half space (right or left) can only be used for training, which also improved the results in temporal order tasks. Hence, in half option, the same level of performance can be achieved using only 50% of the space size.

Due to the rich dynamics being provided by the CA reservoir, multiple methods can be utilized together (ALL-Each-Half, LAST-Each-Half, ..., *f*) which highly reduces the required features by up to 98% in some tasks.

Using All-Each instead of LAST significantly improves ReCA to train the pathological tasks to obtain a zero test error with minimum complexity and required training examples. The minimum required N_{train} examples were obtained in the ALL-Each case as it exploits a large amount of information provided from all time steps with a relatively small feature space after using a few evolution states k instead of all states I. However, LAST is still the best method in a large sequence T and small iterations I, as in random permutation tasks. But, the lack of training examples requires using ALL-Each method.

CHAPTER 4

ReCA VS. FEEDFORWARD ARCHITECTURE AND LOCAL REPRESENTATION

In previous chapter the complexity of ReCA was discussed and we will continue in this chapter by creating *three* other options XOR, Binary, and Gray to reduce the model complexity. In addition, ReCA in feedforward architecture is proposed, also new reservoirs with local representation are proposed to study the effect of the data representation in the reservoir on its function in machine learning.

By reason of comparison, we confine 5-bit task to test ReCA because only this task from the pathological tasks has been used to test other approaches of reservoir computing based on elementary cellular automata [7, 49, 50]. The ordinary 5-bit task is also modified by creating a test set to adapt the machine learning principles for testing the models.

Most parts of this chapter are submitted in Artificial Intelligence Review [85].

4.1 ReCA Implementation

The ReCA framework (Figure 2.1) in Section 2.1 is also used in this chapter with its three stages: Encoding, CA reservoir, and Read-out. The difference in this chapter is that the zero buffers R are only used in the encoding stage as shown in Figure 2.2.

Another difference is creating a new insertion function in the reservoir rather than XOR in previous chapters as demonstrated in Figure 2.5 and explained in Section 2.1.2. The second input with a dimension of L_{in} can be directly inserted into the reservoir by *overwriting* the middle cells in $A_I^{(1)}$ with this input to produce the initial state $A_0^{(2)}$ at second time step. The overwriting (deleting) of $A_I^{(1)}$ middle cells does not lose the first time step information, because the two evolution sides (the propagation in the buffers) of $A_I^{(1)}$ are still conserving the information of the first time step. Therefore, to ensure that, there is no any loss of information; the number of

iterations I should be large enough to propagate the whole input information into the buffers depending on the task nature (size and information position) and ECA rule. Then, the same steps will be repeated up to the last time step to produce the vector $A^{(T)}$ which also memorizes the history of all input sequences. In results section, this option will be called Overwrite and using XOR operation as an insertion function will be called Normal.

4.1.1 Feature Extraction from the Reservoir

As demonstrated in CHAPTER 2 and the results of Section 3.3, the best method of feature extraction is LAST-Each in 5 and 20-bit tasks and ALL-Each in the other tasks. Therefore, in this chapter, LAST-Each in 5-bit task will only be used with expansion ratio *f* and Half option. Also, three new options for feature extraction XOR, Binary, and Gray are created as follows, but to make the context more informative; we will start by Each option:

Each: In this option for each time step *n*; only k states of CA evolution are selected to be used as a feature space. Figure 4.1(a) shows an example of Each option for a certain time step n where k=3 have been used from the feature space $\mathbf{CA}_{out}$ matrix of Figure **3.1** to produce a feature matrix with size of 3×L, thus the dimension of the feature vector $A^{(n)}$ becomes 3L after concatenation. In general, the feature vector dimension in Each is $L_{CA}=(L_{in}+2R)k$.

XOR: the matrix $\mathbf{CA}_{out}$ in Figure 3.1 can be reduced to one row; using bitwise XOR operation for every column of $\mathbf{CA}_{out}$ as shown in Figure 4.1(b) where k=3. Thus, the dimension $L_{CA}=L=Lin+2R$. This case can be considered as an ECA with memory (ECAM); because there is a memory function (XOR operation) added to ECA, but the difference in our case is that this function is only applied after the last iteration, whilst in ECAM the function is applied in all iterations [79].

Binary: Every column vector in $\mathbf{CA}_{out}$ matrix can be represented by converting its binary value to a decimal number with the least significant bit (LSB) at the first row[19]. Figure 4.1(c) shows the Binary option where k=3 in Figure 4.1(a), also it can be used

[19] The most significant bit (MSB) can also be at the first row.

for the whole columns of $\mathbf{CA}_{out}$ matrix in Figure 3.1, the dimension L_{CA}= L. Binary option has also been used in [51] utilizing two ECA rules (one for projection and the other for memory) in the reservoir.

Gray: the binary code is replaced by Gray code (Figure 4.1(d)), the dimension of L_{CA}= L. Gray code is normally used to decrease the noise effect on the binary bits in digital communications and in binary counters [100].

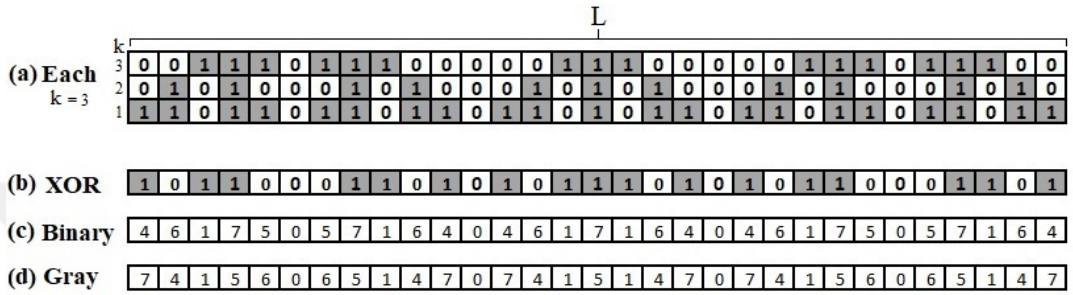


Figure 4.1 Types of feature extraction **(a)** Each; using *3* iterations out of *15* from $\mathbf{CA}_{out}$ matrix in Figure 3.1, i.e., k=3. (**b**) XOR; using bitwise XOR operation for all columns of the matrix in part (a) to produce the feature vector, (c) Binary; converting the binary value of each column of the matrix in part (a) to decimal number, and (d) Gray; using the Gray code instead of the binary code in (**c**) [85].

Notes:

1- Due to the rich dynamics provided by the CA reservoir; (Each, Half, and *f*) options can be used together with any case of XOR, Binary, and Gray.
2- For XOR, Binary and Gray options, the selected iterations k should be greater than or equal 2, because if k=1 they will be similar to Each method with k=1. The advantage of those three options is the low dimension of feature vector L_{CA} even if k is large; still L_{CA}=L, i.e., L_{CA} is independent on the value of k as in Each method where L_{CA}=L.k.

4.1.2 ReCA in Feedforward Architecture

Sequential data can also be handled using feedforward architectures by using a portion of the input to predict an output as in sliding window methods [101] or using the whole input sequence to predict all the output as in [7]. In our work the second method will be adopted via two ways:

All the flattened input sequence is projected to the CA reservoir at once, this method will be called All-ff.

The input of each time step is separately projected to the reservoir then the obtained CA evolution states of all time steps are concatenated together to produce the feature vector that will be used to predict the whole output, this method will be called Each-ff.

In feedforward case the buffer dimension R should be equal to I to hold all CA sequence representations rather than R=I×T for recurrent architecture in Section 2.1.1.1, thus the feature dimension in feedforward architecture is smaller than it in recurrent architecture. But, the advantage of recurrent architecture over the feedforward is the capability to keep a fixed size representation by adjusting the expansion ratio f to a certain length for the feature vector which is similar to the reservoir size N (number of neurons) in ESNs. The CA feedforward architecture is very similar to Extreme Learning Machines ELMs [102, 103], where the random connections in ELM are replaced by the CA reservoir.

4.2 Experiments

In this chapter, ReCA will be tested for long short-term memory capability using only 5-bit memory task which is a part of the pathological synthetic tasks that has been explained in detail in Section 2.3. By reason of comparison, we confine 5-bit task to test ReCA because only this task from the pathological tasks has been used to test other approaches of reservoir computing based on elementary cellular automata [7, 49, 50].

4.2.1 5-Bit Task

The 5-bit memory task is widely used in RNN literature, e.g., [7-9,49,50,88,90,91]. This task is one of the hardest tasks for ESNs in [90], also as reported in [7, 50] it is problematic to solve the 5-bit memory task using feedforward architectures. But, as it will be shown later in Section 4.3.3, it is possible to solve this task using feedforward architecture with some limitations using ECA and even without using it. The weakness of using this task is that; due to its small number of examples (only 32 examples), the whole examples are used to train the model, and then the model is tested using the same 32 examples and finding the error, this approach has been used in [7-9,49,50,

88,90,91], thus the proposed models are not general as it will be explained in the next Section 4.2.2. The details of 5-bit task are demonstrated in Section 2.3.1.1.

4.2.2 Generalized 5-Bit Task

As mentioned in previous section, the 5-bit task has only training set. Thus, the model is only designed for this set and there is no guarantee for the model to solve new (unseen in training) examples that have the same behavior of the training examples, i.e., the model is not general which is a disadvantage for any model. Therefore, to generalize the model and respect the machine learning principles for model evaluation, the dataset should be divided into two sets that generally have different examples; training and testing set. The training set is used for training to calculate the model parameters (the weights of $\mathbf{W}_{\text{out}}$ matrix), and then the model is tested using the input of testing set to predict their output and finding the error between the given output in testing set and predicted one to evaluate the model.

The small dataset is a challenge in machine learning, but nevertheless, we will divide the 32 examples into two sets and even we will search the smallest number of examples that can efficiently train the model to give zero test error. Thus, all the 32 examples of the original 5-bit task are *randomly* shuffled, then the first N_{train} examples are selected for training and then the rest examples will be used for model testing. Hence, the number of testing examples N_{test} can be expressed as follows:

$$N_{test} = 32 - N_{train} \tag{4.1}$$

4.2.3 Training\Testing Stages

After preparing the feature matrix **A** using the various methods of feature extraction mentioned in Section 4.1.1, the features **A** are used in equation (2.13) in Section 2.4.1 for training to find the regression parameters of $\mathbf{W}_{\text{out}}$ matrix, which is used in equations (2.15) or (2.16) in Section 2.4.2 to predict the output. The output is dependent on the task; the predicted output is for testing set output in generalized 5-bit task, or the predicted output is for training set output in 5-bit task.

4.2.4 ReCA Evaluation

After binarization of the predicted output according to the equation (2.17) in Section 2.4.3, the binarized output ($\hat{y}_{(b)test}$ and $\hat{y}_{(b)train}$) will be used to find the model *errors* as follows:

1- testing error=$|\hat{y}_{(b)\text{test}} - \mathbf{y}_{\text{test}}|$ for the generalized 5-bit task, and

2- training error = $|\hat{y}_{(b)\text{train}} - \text{target}|$ for the 5-bit task.

These *errors* represent the number of false bits over all relevant time steps in output bits. The goal of our experiments is to achieve zero test error with minimal complexity, hence the parameters I (CA evolution iterations), R_i (the input expansion to reduce the interference) and *f* (the expansion ratio) should be tuned to their minimum values in order to obtain the shortest length of the feature vector L_{CA}, using the whole 88 equivalent sets of ECA rules that are listed in Table 1.2 excluding the 8 sets of Class I rules; because the evolution of class I rules vanishes after the first iteration in 5-bit task due to the single nonzero in its input at each time step as shown in Figure 2.7. Hence, the ECA rules that will be used are 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 19, 23, 24, 25, 26, 27, 28, 29, 33, 34, 35, 36, 37, 38, 42, 43, 44, 46, 50, 51, 56, 57, 58, 62, 72, 73, 74, 76, 77, 78, 94, 104, 108, 130, 132, 134, 138, 140, 142, 152, 154, 156, 162, 164, 170, 172, 178, 184, 200, 204 and 232 from class II and 18, 22, 30, 45, 60, 90, 105, 122, 126, 146 and 150 from class III and 41, 54, 106 and 110 from class IV, as well as rule 165 (the conjugate of rule 90) and rule 102 (the reflection of rule 60) [104], Thus the total number of used rules in our experiments becomes 82 rules instead of using all the 256 ECA rules.

4.3 Results

The ReCA model has been examined using 5-bit task (32 examples for training) as in [7-9, 49, 50, 90], and using generalized 5-bit task (32 examples for training and testing). Then, ReCA results using several feature extraction methods have been compared with the state-of-the-art results of RC based on CA approaches.

In all experiments; the distractor period T_d=200 time steps, thus the total sequence length becomes T=210. For notation, (I, k, f) indicates the number of total iterations I in the reservoir, k is the selected iterations in Each option and *f* is the expansion ratio unless otherwise described.

In our experiments, we are seeking minimum complexity, i.e., the minimum value of I, k, and *f* that will produce a minimum feature vector dimension L_{CA} in bits to achieve zero test error using minimum training examples in generalized 5-bit Task[20]. The expression '*the model could solve the task*' means that the model could predict correctly (zero error) all the four output bits for every time step for the whole 32 examples, i.e., 4×210×32 bits are correctly predicted in 5-bit task or 4×210×N_{test} bits are correctly predicted in generalized 5-bit task.

4.3.1 5-Bit Task

All proposed methods with several rules and different Wolfram classes could solve the 5-bit task and achieve zero test error as listed in Table 4.1 and Table 4.2 for Normal and Overwrite methods and Table 4.3 and Table 4.4 for XOR, Binary and Gray options with Normal method, i.e., using XOR operator as an insertion. The minimum complexity was attained at (2, 1, 0.26) using rule 165 from class III in Normal method, L_{CA}=224 bits, then Overwrite method using also the rule 165 at (2, 1, 0.33), L_{CA}=282 bits as illustrated in Table 4.2.

Table 4.1 ECA rules that achieve zero test error in 5-bit task using Normal and Overwrite methods with the parameters (I, k, f) and feature vector dimension L_{CA} where I is the number of all CA iterations in the reservoir, k is the number of selected iterations that will only be used for training in Each option, and f is the expansion ratio.

Method	(I, k, f), L_{CA} (Bit)			
	(3, 3, 1)	L_{CA}	(2, 2, 1)	L_{CA}
Normal	106, 30, 45, 105, 165, 150, 2, 10, 34, 38, 42, 46, 56, 74, 130, 138, 162, 170, 184.	3792	105, 165, 15, 34, 38, 42, 162, 170.	1688
Overwrite	106, 30, 45, 90, 102, 105, 165, 150, 2, 10, 11, 15, 34, 38, 42, 43, 46, 56, 74, 130, 138, 154, 162, 170, 184.	3792	90, 38, 42, 154, 170.	1688

[20] In 5-bit task, all the 32 examples should be used.

Table 4.2 ECA rules that achieve zero test error in 5-bit task using **Normal** and **Overwrite** methods with the parameters (I, k, f) and feature vector dimension L_{CA}. The minimum dimension L_{CA} is in bold numbers. Note: For Half option, the right side was used in rule 15, any side from the both can be used in rules 90, 165 and the left side was used in the other rules.

Method	(I, k, f), L_{CA} (Bit)					
	(2, 1, 1) Half	L_{CA}	(2, 1, 0.5)	L_{CA}	(2, 1, f)	L_{CA}
Normal	165, 15, 34, 42, 162, 170.	424	15, 34, 42, 162, 170.	424	165, f=0.26	22**4**
Overwrite	90, 42, 170.	424	42, 170, 90, f=0.55	424 466	165, f=0.33	282

There is an obvious *risk* of information loss when using Overwrite with *two* iterations (I=2) and *one-way flow* rules 38, 42 and 170 (see Figure 4.2) for the input with 4 bits dimension as in 5-bit task; the last two inputs u_2 and u_3 in Figure 2.7 will be deleted after using the Overwrite insertion function in the reservoir for the next time-step, the question why did Overwrite method give good results as listed in the above tables? Because the input memory pattern is in the first two inputs u_0 and u_1 thus, they will be reserved in the buffers R using two iterations and then they can be repeated at the output as demonstrated in Figure 2.7 in the last time steps. The nonzero in the third input u_2 is not important for the output it is only a distractor, but the cue signal in forth input u_3 is important because after its arrival the output will repeat the input pattern, thus the importance here is 'when does the cue signal arrive?', the cue signal still appears in the CA reservoir state at the time step $T - 5$ as shown in Figure 2.7, then it is followed by the input memory pattern in the output. Hence, the Overwrite method, with the nature of 5-bit task, could save the necessary task information and provide good results.

But, the using of Overwrite method generally requires the number of iterations I ≥ (dimension of L_{in}) / 2 for ECA rules with two-way flow or I ≥ (dimension of L_{in}) for ECA rules with one-way flow, Figure 4.2 depicts some examples of such rules.

In XOR, Binary and Gray, the minimum complexity was reached at (2, 2, 0.5) and (2, 2, 1) Half where L_{CA}=424 bits; using the rules (15, 34, 38, 42, 162, and 170) from class II and 165 from Class III for only Half option as listed in Table 4.4.

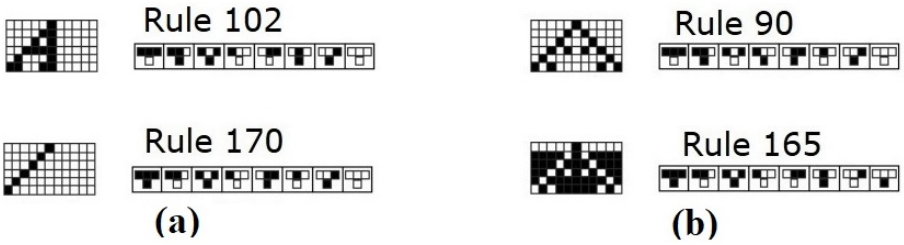


Figure 4.2 Information flow in ECA for five iterations space-time diagram of different rules for a single non-zero initial state with one iteration of a center cell for each rule: (**a**) One-way and (**b**) Two-way [85].

Notes:

1- In the XOR, Binary and Gray options, k=1 was not used; because in this case, those three options will be as Normal method when k=1 which was already listed in Table 4.2.
2- For simplification, the insertion function XOR has only been used in the XOR, Binary, and Gray options. The overwrite function will be used in future work.

Table 4.3 ECA rules that attain zero test error in 5-bit task using Normal method for XOR, Binary and Gray options with the parameters (I, k, f) and feature vector dimension L_{CA}.

Method	(I, k, f), L_{CA} (Bit)			
	(3, 3, 1)	L_{CA}	(3, 2, 1)	L_{CA}
XOR	2, 10, 38, 74, 130, 138.	1264	106, 2, 10, 15, 24, 34, 38, 42, 46, 56, 74, 130, 138, 152, 162, 170, 184.	1264
Binary	106, 165, 2, 10, 34, 38, 42, 46, 56, 74, 130, 138, 162, 170, 184.	1264	106, 165, 2, 10, 24, 34, 38, 42, 46, 56, 74, 130, 138, 152, 162, 170, 184.	1264
Gray	106, 90, 165, 2, 10, 34, 38, 42, 46, 56, 74, 130, 138, 162, 170, 184.	1264	106, 90, 165, 2, 10, 15, 24, 34, 38, 42, 46, 56, 74, 130, 138, 152, 162, 170, 184.	1264

In the cases where *f=0.5* and Half option with *f=1* (not less) are almost the same because the periodic boundary condition has been utilized with one-way rules (15, 34, 38, 42, 162 and 170) as represented in Figure 4.4. Hence, the same features are obtained in the 2 cases; because it is just a columns permutation (as described in Figure **4.3**(a), (b), and (c)) where the columns permutation does not affect the regressor results in the read-out stage.

Table 4.4 ECA rules that achieve zero error in 5-bit task using Normal method for XOR, Binary and Gray options with the parameters (I, k, f) and feature vector dimension L_{CA}. The *minimum* dimension L_{CA} is in bold numbers. *Note:* For Half option, the right side was used in rule 15, any side from the both can be used in rules 165 and the left side was used in the other rules.

Method	(I, k, f), L_{CA} (Bit)			
	(2, 2, 1)	L_{CA}	(2, 2, 0.5) and (2, 2, 1) **Half**	L_{CA}
XOR	34, 162.	844	34, 162.	424
Binary	165, 15, 34, 38, 42, 162, 170.	844	165 **Half,** 15, 34, 38, 42, 162, 170. 165, f=0.66	424 560
Gray	165, 34, 38, 42, 162, 170.	844	34, 38, 42, 162, 170.	424

But the difference is still existing in the rules 90 and 165 where they have two sides of propagation. Therefore, the obtained information from both sides is different. Moreover, for rule 165 with Half option; *f* can be less than 1 (*f* = 0.9 experimentally in 5-bit task) due to its high distributedness that has also been proved for rules 90 and 150 in [8, 9].

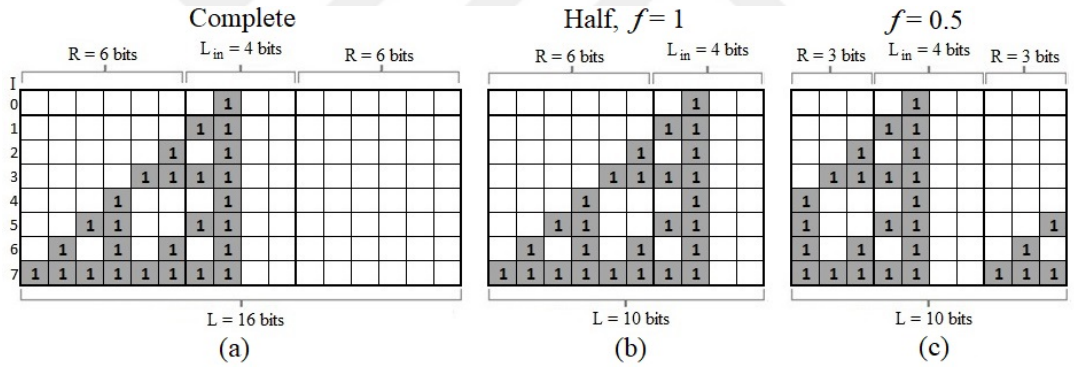


Figure 4.3 Feature space for a one-way rule: (**a**) Complete; using the whole space (Both sides and *f* = 1), hence the largest length L=16 bits, (**b**) Half, *f* =1; Using only the left side but *f* should be equal to 1 to conserve the whole information. In this case, L=10 bits, and (**c**) *f* = 0.5; Using both sides for R = 3 bits rather than 6 bits in (**b**). The first three columns in (**b**) are transferred to last three columns in (**c**) after using periodic boundary condition, thus (b) and (c) are identical in the regressor. White squares represent zeros [85].

4.3.2 Generalized 5-Bit Task

As mentioned in Section 4.2.2, the training and testing sets are selected *randomly* from all the 32 examples of the 5-bit task. Hence, there are a lot of different samples, therefore to get stable estimates of model performance; Monte Carlo cross-validation

technique was applied where the algorithm run should be applied multiple times[21] [105]. The single run (trial) is said to be successful (zero test error) if the ReCA could find the correct binary value for all the 4 output bits for all time steps for whole testing examples N_{test}, this means $4\times210\times N_{test}$ bits are correctly predicted for one trial, to *solve* the generalized 5-bit task, the correct prediction should be for all the 100 trials.

ReCA could solve the generalized 5-bit task using a very small number of training examples N_{train}=2 or 3 to achieve zero test error as listed in Table 4.5. The minimum complexity was at (2, 1, 0.5) using Overwrite method with the rules 42 and 170 from class II where N_{train}=3 examples and L_{CA}=424 bits. But, for N_{train}=2 examples, the task becomes harder. Therefore, the feature vector dimension L_{CA} increases from 424 to 634 bits and the number of CA iterations I increases from 2 to 3 to obtain zero test error using Overwrite and XOR methods with the rules 42 and 170 from class II as listed in Table 4.5. Based on the same table, the increasing of the values of ReCA parameters I, k and/or *f* makes more rules can achieve zero test error but of course with larger complexity.

Table 4.5 ECA rules that Succeeded to obtain zero test error using several methods for feature extraction with *two* and *three* examples for training and 100 trials. The *minimum* dimension of L_{CA} is in bold.

Method	N_{train}	(I, k, f)	ECA rule	L_{CA} (Bit)
Normal	2	(4, 1, 0.5)	106, 2, 10, 15, 34, 38, 42, 74, 138, 162, 170, 184.	844
Overwrite[22]		(3, 1, 0.5)	42, 170.	634
XOR[23]		(3, 2, 0.5)	42, 170.	634
Binary		(4, 2, 0.5)	106, 34, 42, 162, 170.	844
Gray[24]		(4, 2, 0.5)	34, 162.	844
Normal	3	(3, 1, 0.5)	106, 34, 42, 162, 170.	634
Overwrite		(2, 1, 0.5)	42, 170.	424
XOR, Binary, and Gray		(3, 2, 0.5)	106, 34, 42, 162, 170.	634

[21] The experiments have been repeated 100 times, i.e., N_{trials} =100 runs (trials).
[22] Rules 106, 34 and 162 gave only 1 error in 100 trials.
[23] Rules 106, 34 and 162 gave only 2 errors in 100 trials.
[24] Rules 106, 42 and 170 gave only 1 error in 100 trials.

4.3.3 CA Feedforward Architecture

In CA feedforward architecture, there are two methods to project the sequence input into the reservoir All-ff or Each-ff as explained in Section 4.1.2. The most of ECA rules except the rules 23, 72, 104, 200 and 232 could solve the 5-bit task using those two methods with a small value of the parameters (I, k) = (1, 1). All-ff outperforms in the complexity with feature vector dimension L_{CA}=842 bits while L_{CA}=1260 bits for Each-ff. For the generalized 5-bit task, Each-ff outperforms a little bit where the model achieves zero test error using only 10 training examples while 11 examples are required for training to achieve zero test error in All-ff model.

The small number of iterations I in the previous results (recurrent and feedforward) also the shift property in the most success rules as shown in Figure 4.4 for a single nonzero initial state as in our case for 5-bit task lead us to test the local representation in 5-bit tasks.

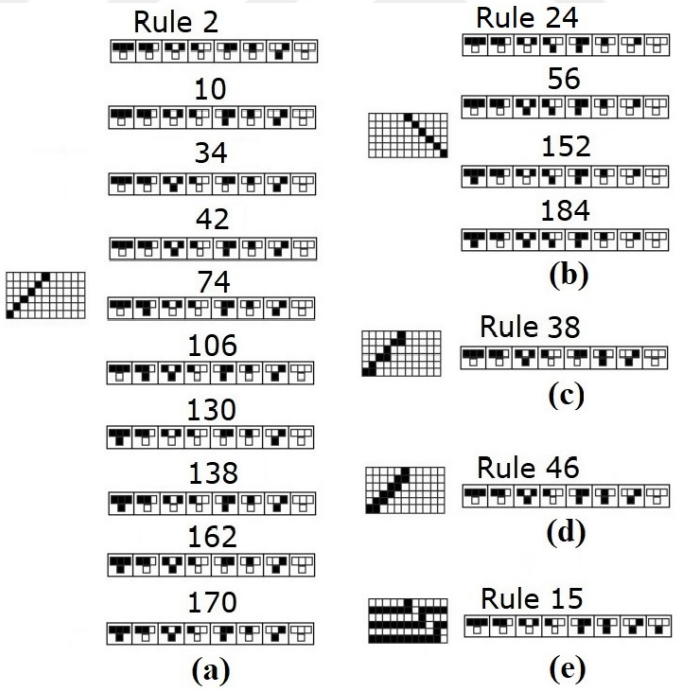


Figure 4.4 Shift rules; five iterations space-time diagram of different ECA rules for a single non-zero initial state with one iteration of a center cell for each rule. **(a)** left shift, **(b)** right shift, **(c)** left shift single and double bits, **(d)** left shift double bits, and (e) inverted right shift. ***Note:*** The shift is independent on the initial state only for two rules 170 and 15 where their Boolean expressions are $x_i(n+1) = x_{i+1}(n)$ and $x_i(n+1) = \bar{x}_{i-1}(n)$ respectively, where $\bar{x}$ is the inverse of x [85].

It is an encroachment for using the word 'shift' for these rules (Figure 4.4) because they are shift rules only at first time-step, but after insertion the second time-step to the reservoir, there is no guarantee to stay as shift rules, it is dependent on the nonzero positions in the new initial state at second time step and so on for the other time steps. Also, we should note that the local representation is applied only in the reservoir, but the model still has a computational power from the regressor in the read-out stage to solve the tasks.

4.3.4 Local Representation Models

In local representation, there are also feedforward and recurrent architectures, in the local feedforward the input sequence is flattened (vectorization) and is *directly* used as features for training to predict the whole output, it will be called (without ***CA***). But, in local recurrent, the sequence input is memorized consecutively step by step as shown in Figure 2.6. Thus, the insertion function is similar to the *stack* memory, the first row of feature space which has only the first time step input is used to predict the output at first time step and so on up to last time step consisting of all input time steps as illustrated in Figure 2.6. This model has been used in Section 2.2.2 and [8] for comparison with ReCA to study the required distributedness and computational to solve sequence tasks and is called *stack* reservoir. The dimension of feature vectors in stack reservoir is dependent on the input dimension L_{in} and the sequence length T as shown in Figure 2.6. Thus, it is not a fixed length model and from this point of view; it is like feedforward, which is problematic for large T and L_{in}.

Surprisingly, the local feedforward and stack reservoir could solve the 5-bit task with feature vector dimension L_{CA}=840 bits for both; but, this vector is used to predict the whole output for local feedforward (without ***CA***) while for stack representation it is only utilized to predict the output in a single time step n. The local feedforward (without ***CA***) and stack reservoir could solve the generalized 5-bit using 11 examples and only 3 training examples respectively.

4.3.5 Comparison with other Approaches

The model complexity is a useful metric to evaluate different models [95]. Therefore, the feature vector dimension L_{CA} (bits) will be used to compute the model complexity

for different RC based on CA approaches (a snapshot of this approaches is presented in Section 1.3) that have been used 5-bit task. Table 4.6 shows the best results of recurrent architectures from those approaches.

Table 4.6 The minimum complexity results for 5-bit task where T_d=200; using several approaches that utilize recurrent architecture of RC based on CA except stack reservoir which is used local representation instead of CA. The results are listed in ascending order, i.e., the best result is on the top. For any details, please see the appropriate reference in the 2nd column.

Method	Reference	L_{CA} equation	The values of L_{CA} Parameters	L_{CA} (Bits)	Successful Rules
Normal	Chapter 4	(Lin+2(f.I.T)).k	(4+2(0.26×2×210))×1	224	165
Overwrite	Chapter 4	(Lin+2(f.I.T)).k	(4+2(0.33×2×210))×1	282	165
Non-Uniform CA	[49]	(C.Lin).R.I	(10×4)×4×2	320	90 with 165
ReCA LAST-Each-Half	[9]	(Lin+(f.I.T)).k	(4+(0.15×4×210)) ×3	390	150
XOR	Chapter 4	(Lin+2(f.I.T))	(4+2(0.5×2×210))	424	34, 162
Binary	Chapter 4	(Lin+2(f.I.T))	(4+2(0.5×2×210))	424	15, 34, 38, 42, 162, 170
Gray	Chapter 4	(Lin+2(f.I.T))	(4+2(0.5×2×210))	424	34, 38, 42, 162, 170
ReCA **LAST**	[8][25]	(Lin+2(f.I.T)).I	(4+2(0.1×4×210))×4	688	90
Stack Reservoir	Chapter 4	$T.L_{in}$	210×4	840	
Deep learning CA Single reservoir	[50]	L_d.R.I	40×8×8	2560	90
Deep learning CA Two reservoirs	[50]	$(L_{d1}.R_1.I_1)+(L_{d2}.R_2.I_2)$	(40×8×8)+(30×8×8)	4480	90 and 90
CA Based Feature Expansion and RC	[7]	L_{in}.R.I	4×38×32	4864	150

For the generalized 5-bit task, there are no other results except our results in Section 4.3.2. Therefore, to compare our results with another model, the ESN experiments in [90] have been repeated applying the generalized 5-bit task with three

[25] In this reference T_d=1000. For that, the experiment has been repeated for T_d=200.

levels of effort/expertise (Blind, Basic, and Smart) [90], the results are listed in Table 4.7. Finally, Table 4.8 shows the minimum training examples that are required to solve the generalized 5-bit task using several methods with different architectures.

Table 4.7 Minimum training examples for the generalized 5-bit task where $T_d=200$ to attain zero test error using echo state networks ESNs with three levels of effort/expertise, where N is the reservoir size, α is the leaking rate and ρ is the spectral radius. For more details please see [90].

Method		N	α	ρ	N_{train}
ESNs	Blind	2500	1	1	28
	Basic	500	1	1	22
	Smart	200	1	1	10

Table 4.8 Minimum training examples for the generalized 5-bit task where $T_d=200$ to achieve zero test error using different methods with their parameters.

Architecture	Method	N_{train}	(I, k, f)
Recurrent	Normal	2	(4, 1, 0.5)
	Overwrite	2	(3, 1, 0.5)
	XOR	2	(3, 2, 0.5)
	Binary and Gray	2	(4, 2, 0.5)
	Stack reservoir	3	
Feedforward	All-ff	11	(I, k) = (1,1)
	Each-ff	10	(I, k) = (1,1)
	Without CA	11	

ESNs with the highest level of effort (Smart) [90] could achieve zero test error using 10 training examples with N=200 neurons in the reservoir while for basic and blind the value of training examples increased to 22 and 28 respectively. But, all ReCA methods in this paper could solve the generalized 5-bit task using only 2 or 3 training examples with a very low effort 3 or 2 iterations, using 634 bits or 424 bits as a feature vector instead of 6400 bits in the best case (smart) of ESNs for 32 bits floating point.

4.4 Discussions

Three methods are provided to extract the features from CA evolution states in ReCA model. 5-bit and generalized 5-bit tasks have been used in order to compare and

evaluate these methods with each other and with the state-of-the-art of other approaches in RC based on CA domain.

The presented results reveal that some of ECA rules in classes II, III and IV with all proposed methods could solve the 5-bit task with different levels of complexity. The minimum complexity was obtained using Normal method and rule 165 from class III with feature vector length of 224 bits due to its high distributedness, compared with shift rules, which enabled it to collect large information in a small dimension. Increasing the model complexity by increasing the number of CA iterations I increases the number of rules that could solve the 5-bit task due to the increasing of computational power as proved in [7].

All proposed methods with some of class II rules and rule 106 from class IV could solve the generalized 5-bit task with different levels of complexity. The minimum complexity was achieved using Overwrite and XOR methods with rules 42 and 170 from class II and feature vector with dimension of 634 bits using only *two* examples for training, while for the *three* training examples the minimum complexity was obtained using Overwrite method with rules 42 and 170 from class II with feature vector dimension of 424 bits. Increasing the training examples provides the model more information which enables more ECA rules to solve the generalized 5-bit task with less complexity, also increasing the model complexity by increasing *I* increases the model computational power to use less training examples and more rules can also solve the task. Hence, there is a trade-off between complexity and the required training examples, but we should be careful of the overfitting in complicated (large complexity) models then further of training examples will be required to generalize the model. That is why the minimum complexity (L_{CA}=224 bits) has been reached for the 5-bit task where all the 32 examples have been used for training, i.e., the model is not general.

ReCA could solve the generalized 5-bit task using only *2* training examples, which is the lowest limit number for training for any model. Thus, we can argue that ReCA can be used in the cases of lack of examples, especially the same result was obtained in [8] for all the eight pathological tasks. But, ReCA must be tested for large space of tasks to prove such claim.

In most literature, the researchers focus on classes III and IV of ECA due to the chaotic behavior of class III which is used in cryptography and random number generators, and the complex behaviors of class IV with its computational universality. But, as it has demonstrated in this chapter, class II ECA rules have presented the best results in the generalized 5-bit task. Hence, class II rules should be given more prominence in the next studies.

The combination of all iterations I of CA evolution states using XOR, Binary and Gray options provides an efficient feature vector which could solve the 5-bit memory tasks. to take advantages of XOR, Binary or Gray options over Normal and Overwrite methods the parameter k should be $k \geq 2$ because in this case, the XOR, Binary and Gray options change the parameter *k* to 1, which reduces the dimension of feature vector that will be used in the read-out stage. The reservoir size for all methods is equally and depends on I, k and f; the difference is in the feature vector dimension L_{CA}[26] to be used in the regressor in order to find the pseudo-inverse, that implies the most expensive computational part in the model.

For 5-bit task, the best results have been obtained using rule 165 which outperformed its conjugate rule 90 due to the increasing of nonzero bits (Figure 4.2(b)), which decreases the mean pairwise correlation of the CA features for different inputs as in [7], i.e., improving the computational power [32]. But, both rules 165 and 90 could not solve the generalized 5-bit task due to their complex behavior (class III), which is introduced the model to overfitting, to overcome this problem; an excess of training examples are required or reducing the model complexity by choosing more simple rules, e.g., shift rules 42 and 170 as we have done in Section 4.3.2.

Only, class II ECA rules could solve the generalized 5-bit task with minimum complexity. But, due to the relative simplicity of their behavior and to avoid the underfitting; the model needs more iterations *I* for a small number of training examples or needs more training examples for a small number of iterations. We should not be confused with rule 106 which could solve the generalized 5-bit task, yet it is from class

[26] For Normal and Overwrite, L_{CA} depends on I, k, *f*, but it depends only on I, *f* for XOR, Binary and Gray.

IV; because its behavior is similar to the shift rules from class II for a single nonzero input as illustrated in Figure 4.4.

Only 82 rules out of all the 256 ECA rules have been utilized in this chapter and these rules have been selected from the equivalent sets of ECA rules [71], but it is not enough; because as an example, the rule 90 and its conjugate rule 165 (also rule 60 and its reflection rule 102) are in the same equivalent set in [71] but experimentally they have obtained different results. Therefore, the whole ECA rule space has to be exploited in future work.

The stack reservoir is a recurrent architecture because its states in the reservoir memorize the input information gradually time-step by time-step, but on the other hand, it has not a fixed length, so it is like feedforward architecture.

In spite of good results obtained from the feedforward models for 5-bit task, there are some points should be discussed here: feedforward is conceptually not plausible; because the subsequent (new) inputs are used to predict some previous (old) output. So, the provided information is increased for prediction, that is why it gives good results. Feedforward architecture needs more training examples 10 or 11 to solve the generalized 5-bit task while the recurrent architecture requires 2 or 3 as listed in Table **4.8**. Finally, the Stack and feedforward models are problematic for large input dimension L_{in} and/or large sequence T; because they have not a fixed length, i.e., their feature vector length increases proportionally with L_{in} and T. Therefore, ReCA is still the best choice for sequence learning.

The comparison in Table 4.6 indicates that the best results of all RC based on CA approaches have been achieved using the linear (additive) rules (165, 90 and 150) or shift rules (15, 34, 38, 42, 162 and rule 170 which is also additive) that provide lossless injection of input at each time step, i.e., maximizes one-to-one correspondence between input sequence and the reservoir activity due to the sequence. Moreover, the additive rules can be represented as linear functions modulo two, thus these rules allow to compute independently the evolution for different initial states, then the results can be combined by simply adding which significantly simplifies the hardware implementation of these rules.

The comparison in Table 4.6 demonstrates that the feature expansion using zero buffers is better than using random permutation; due to the natural information diffusion into the CA reservoir there is no information loss, but the random permutation may need a lot of permutations (large size) to ensure there is no information loss. Moreover, the permutations increase the random interference between the input cells into the reservoir while the model becomes more robust using the zero buffers; because the evolution states of the input obey a certain rule. This interpretation agrees with our results where the Normal and Overwrite methods outperform a little bit the other options (XOR, Binary and Gray); because in Normal and Overwrite methods the feature vector is the pure CA evolution states without any intervention but in XOR, Binary and Gray options, the feature vector is modified by an operator. This interpretation is, of course, valid for the used 5-bit memory tasks. Thus, these methods should be tested using other types of tasks, then study which method is the best for every task as will be done in future work.

In ReCA, there is no parameter selected randomly as in ESNs or the other approaches of RC based on CA [7, 49, 50], that's why we did not repeat our experiments multiple times in 5-bit task as in [7, 49, 50, 90], but for the generalized 5-bit task we have repeated the experiments 100 trials due to the random selection of training examples from the whole dataset, not from ReCA model.

ReCA has outperformed ESNs for the 5-bit task, as demonstrated in [7, 8]. For the generalized 5-bit task, ReCA also outperforms ESNs where ReCA could solve the task using only *two* training examples with a very simple model but *smart* ESNs which is the most complicated model could solve this task using *ten* training examples while for the lower complexity models *basic* and *blind* ESNs, they require *22* and *28* training examples respectively to solve this task.

CHAPTER 5

NONBINARY AND STATIC TASKS

After the promising results of ReCA in previous chapters using pathological synthetic tasks, that are binary tasks, ReCA will be tested using real and nonbinary tasks in order to generalize its applications. We will start with a simple signal classification task in Section 5.3 then continue with the Japanese vowels task [109] (Section 5.4), which is multidimensional dynamic pattern recognition. Finally, tough ReCA is designed for sequence learning; it will be tested using static IRIS data set [118] in Section 5.5. All those tasks are nonbinary dataset, but ReCA deals with only binary data. Therefore, the nonbinary data have been binarized using one hot encoding in Section 5.2.

5.1 ReCA Implementation

ReCA is implemented as in Section 4.1 using the option Each as a feature extraction from the CA reservoir as in Section 4.1.1, where I is the total number of iterations in the reservoir and k is the number of selected iterations, that is used in training to compute the regression parameters of $\mathbf{W}_{out}$ matrix, which is then used to predict the output.

5.2 One Hot Encoding

All tasks that will be used in this chapter are nonbinary, but ReCA deals only with binary data. Therefore, the one hot encoding, due to its simplicity, will be used to binarize such data[27].

One hot encoding is widely used to encode the categorical variables (integer numbers) to orthogonal and equidistant categories, which agrees with classical intuitions about nominal categorical variables [106]. But, the one hot encoding is problematic in large data because each variable is represented by one bit.

[27] Other binarization methods can be used such as Binary or Gray code.

In one hot encoding, each decimal value is represented by ‘1’ which is located according to its decimal value in zero array as illustrated in Table 5.1.

Table 5.1 One hot encoding: Decimal numbers are represented by one hot encoding.

Decimal No	One hot representation						
	1	2	3	….	n	….	max
1	1	0	0	. 0 .	0	. 0 .	0
2	0	1	0	. 0 .	0	. 0 .	0
3	0	0	1	. 0 .	0	. 0 .	0
⋮	0	0	0	1	0	0	0
n	0	0	0	. 0 .	1	. 0 .	0
⋮	0	0	0	0	0	1	0
max	0	0	0	. 0 .	0	. 0 .	1

5.3 Sin/Square Classification Task

Sine/Square wave classification task is a time-series binary classification problem. In this task, the input is a random concatenation of sine and square waves (blue waves in Figure 5.1). ReCA should be capable to correctly classify whether the input wave is sine or square at every time step. The output is represented by ‘1’ for square wave and ‘0’ for sine wave, which is represented in Figure 5.1 by the red points.

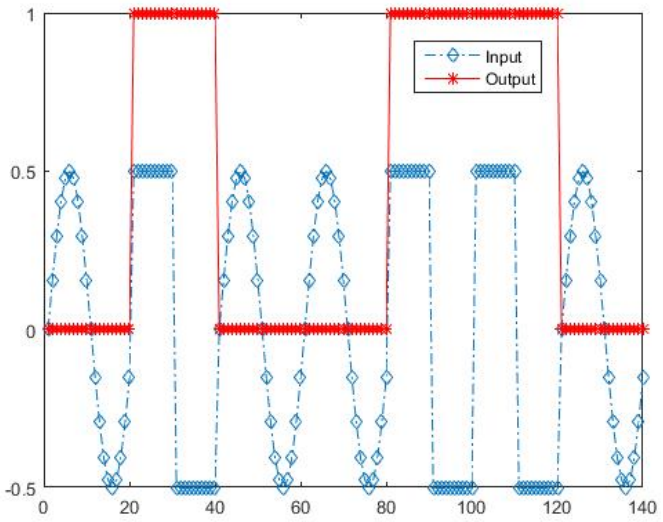


Figure 5.1 Sine/Square wave classification Dataset.

At first glance, this task seems very simple, but the problem is the nonlinearity at the input values -1 and 1 where the output is either sine or square wave. Hence, the model

should remember what is before these values (-1 and 1) to decide the wave type, i.e., the model should have a memory.

5.3.1 Input Binarization

The sine wave is a continuous signal, therefore it should be discretized and produce a specific number of samples, 20 as an example, from a sine wave with an amplitude of 0.5 as illustrated in the second row of Table 5.2, then it will be binarized using one hot encoding with 11 bits as listed in Table 5.2. For square wave, there are only two levels -0.5 and $+0.5$ that can be represented by one hot encoding as follows: (00000000001) for -0.5 and (10000000000) for $+0.5$.

Table 5.2 Binarization of 20 samples of a sine wave with an amplitude of 0.5 using 11 bits one hot representation.

Sample No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Value	0.00	0.15	0.29	0.40	0.48	0.50	0.48	0.40	0.29	0.15	0.00	-0.15	-0.29	-0.40	-0.48	-0.50	-0.48	-0.40	-0.29	-0.15
MSB	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
One hot Representation	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
LSB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

5.3.2 Results

The ReCA performance on this task is essentially perfect where ReCA could correctly predict 500 waves, i.e. 10000 bits, in the test set with very low effort; using only one state k=1 out of 4 iterations with rule 106 using only 10 waves for training. Whilst, for the other three rules (54, 41, 90) ReCA requires more iterations I and training examples as listed in Table 5.3. For comparison, Table 5.3 shows that ReCA outperforms, with a significant gap, the results in [51], which is also used reservoir computing based on cellular automata with another approach.

The model performance can also be measured by the normalized mean square error (NMSE) as follows:

$$NMSE = \frac{\frac{1}{T}\sum_{n=1}^{T}(y(n)-\hat{y}(n))^2}{var(y)} \quad (5.1)$$

where $y(n)$ is the desired output in test set, $\hat{y}(n)$ is the predicted output, $var(y)$ is the variance of the desired output $\mathbf{y}$, and T is the sequence length.

Table 5.3 ReCA parameters to solve the Sine/Square task and comparison between our results with 4 ECA rules (top) and the model in [51], which is also used RC based on CA (bottom).

Method	ECA rule	Class	I	k	N_{train} (waves)	N_{test} (waves)	All test bits
ReCA	106	IV	4	1	10	500	10000
	54	IV	5	1	100	500	10000
	41	IV	7	1	220	500	10000
	90	III	15	15	200	500	10000
[51]	Multiple		80	2	200	200	4000

ReCA could solve the task with NMSE=1.6×10^{-4}. Hence, ReCA outperforms the methods in [40] using an optoelectronic reservoir and [107] using a photonic reservoir where the NMSE was 1.5×10^{-3} for both methods. Moreover, the NMSE can be improved to 2×10^{-5} by increasing the number of training waves to 20 instead of 10 waves as in Table 5.3.

5.4 Japanese Vowels Task

Japanese vowels task is widely used in machine learning to test models for multidimensional dynamic patterns recognition. The first presentation of this task was by Kudo et al. [108] to validate their multidimensional curve classification system and is available at the UCI Knowledge Discovery in Databases Archive [109], then it was used in, e.g., [110-114]. It is a real-world dataset consisting of samples of speakers saying the Japanese vowels "ae" successively. This dataset is a time series of different length (from 7 to 29) depending on the duration of the articulation; comprised of multiple feature dimensions 12 inputs (12 Linear predictive coding (LPC) cepstrum coefficients [115]), each of which is spoken by one of nine males; Figure 5.2 shows a sample of the 12 LPC coefficients in Japanese vowels task. The dataset consists of 640 time series; 270 for training (30 for each speaker), and the rest 370 time series are used for testing that contains several samples for the 9 speakers.

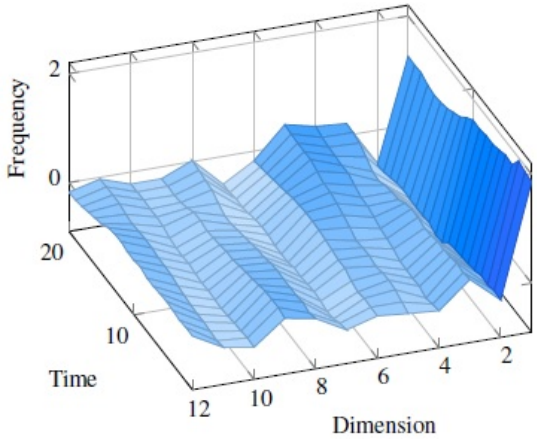


Figure 5.2 The 12 LPC coefficients for a sample that has 20 time steps in Japanese vowels task [116].

5.4.1 Results

One hot encoding has been used for representing both the input and output. Since the task has 9 speakers, therefore 9-bit output array should be used. After completing the ReCA algorithm as in last chapters and obtaining the predicted output; the maximum value in the predicted output array was set to be '1' and the other values were set to be '0's. Then converting the one hot output array to a decimal value and compare it with the correct output (whose male does speak?) to find the training and testing error.

Table 5.4 shows the results of Japanese Vowels Task. The minimum test error can be achieved by ReCA is 3.5%, whereas the-state-of-the-art machine learning methods achieve between 1.1% and 2.7% in percentage test error [114]. Thus, ReCA results are not far from these results. The small difference in our results due to the binarization error, because all the other methods deal directly with decimal numbers.

Table 5.4 Japanese Vowels Task: Results obtained from ReCA. Where I is the number of CA iterations in the reservoir, k is the selected evolution states in the reservoir to be used for prediction in the read-out stage.

ECA Rule	Class	I	k	No of errors in testing set	Test error %
41	IV	2	1	13	3.5%
60	III	2	2	14	3.8%
126	III	1	1	15	4.1%

5.5 IRIS Task

The aim of IRIS task is to classify iris flowers among 3 types (classes) Setosa, Versicolor or Virginica; from measurements of length and width of their sepals and petals in cm's[28]. Each class consists of 50 examples see [117] and can be obtained from [118] in UCI Knowledge Discovery. Hence, there are 150 examples, that can be divided as follows: 105 (70%) for training and 45 (30%) examples for testing.

5.5.1 Feedforward and Recurrent Architecture

In previous tasks, ReCA deals with dynamic dataset. But, in this section, ReCA is tested in static learning. Therefore, two architectures are proposed:

1. Feedforward Architecture: all the inputs are concatenated in one array, which is used as an initial state for the CA reservoir, so we have only one time step, but with a large dimension.
2. Recurrent Architecture; In this case, the inputs are considered as time steps. Thus, the first input is used as an initial state for the CA reservoir in first time step, then inserting the second input using XOR as an insertion function and so on up to the last input.

Using time-dependent data into a feedforward architecture has been used in the literature, e.g., Section 4.1.2. But, using time-independent data into a recurrent architecture, it was never used as far as I know.

5.5.2 Results

The binarization of IRIS dataset is similar to that of Japanese Vowels task, but the output will only be 3 bits because there are only 3 classes. the IRIS dataset is static; thus, the two architectures feedforward and recurrent are used as explained in the previous section.

[28] IRIS is time-independent dataset.

Table **5.6** indicates that the results of recurrent architecture have reached zero test error and outperform the feedforward architecture in Table 5.5 where the best results are 14 out of 45 (31.11%).

Table 5.5 IRIS Task. Feedforward architecture results.

ECA rule	Class	I	k	No of errors in test set	Test error %
126	III	3	1	14	31.11%
105		7	2		
150		7			
30		8			
		8	3		
		2	1	15	33.33%
126		3	3		

Table 5.6 IRIS Task. Recurrent architecture results.

ECA rule	Class	I	k	No of errors in test set	Test error %
110	IV	4	3	0	0%
110			4		
122	III	5	3		
105		6	5		
150			4		
150			5		
60		7	3		

The ReCA performance in recurrent architecture outperforms the related work that uses RC based on CA with different approach reported in [51], where the test error was 3.8% with very high effort compared with ReCA in this chapter, which has a zero test error with a very low effort as listed in Table 5.7.

Table 5.7 IRIS Task: Comparison between ReCA and the related work that uses RC based on CA with a different approach in [51].

Method	ECA Rule	Class	I	k	N_{train}	N_{test}	Test error%
ReCA	110	IV	4	3	105	45	0%
[51] (2017)	158	II	> 70	2	112	38	3.8%

Table 5.8 shows that four classifiers in [127] (decision tree, multilayer perceptron FNNs, Naïve Bayes and supporting vector machines (SVMs) multiclass classifier) couldn't reach zero error. These classifiers have been implemented using WEKA tools which is an open source data mining software issued under General Public License [128].

Table 5.8 IRIS dataset results for various classification algorithms using WEKA tools [127].

Method	Error%
Decision tree	50%
Multilayer Perceptron	2.7%
Naïve Bayes	4%
Multiclass Classifier SVM	4%

5.6 Discussions

In the previous three chapters, ReCA could solve the pathological synthetic tasks, that are binary and time-dependent data set. In order to study the powerful of ReCA; real and nonbinary tasks (signal classification and Japanese vowels) are proposed to examine ReCA. Also, to generalize the applications of ReCA, it has tested using a time-independent task (IRIS dataset).

The sine/square classification task has been used where the continuous input has been discretized then binarized using one hot encoded. ReCA performance was perfect and outperforms the related work that uses RC based on CA with different approach reported in [51], also ReCA results have the smallest NMSE comparing to the state-of-the-art results.

ReCA could solve real and nonbinary task like Japanese vowels which is a multidimensional dynamic task with competitive results 3.5% test error compared with 1.1% to 2.7% test error in the-state-of-the-art machine learning methods. The small difference in our results due to the binarization error, because all the other methods deal directly with decimal numbers.

Finally, ReCA was also tested by a real, nonbinary and static task (IRIS Task), to solve this task two models are proposed (Feedforward) and (Recurrent). The recurrent architecture could achieve a zero test error whilst feedforward could not solve this task with zero error due to the high distributedness which is provided by the recurrent architecture as proved in [7, 8]. In IRIS dataset, ReCA outperforms the decision tree, multilayer perceptron feedforward ANNs, Naïve Bayes and supporting vector

machines (SVMs) multiclass classifiers where ReCA achieves zero test error however the other classifiers could not achieve zero training error. Also, ReCA outperforms the related work that uses RC based on CA with different approach in [51], where the test error was 3.8% with very high effort compared with ReCA which reaches zero test error with a very low effort.

CHAPTER 6

ReCA in QUESTION ANSWERING

After the ReCA success in (Real and Artificial), (Binary and Nonbinary), and (Static and Dynamic) tasks in previous chapters, it should be tested by a more difficult and recent task as a question answering (QA) task. The QA 20 bAbI task from Facebook [120] is a very hard and complex natural language processing (NLP) and requires an understanding of the meaning of a text and the ability to reason over relevant facts.

Most tasks in NLP can be considered as a QA problem: high level tasks like machine translation (What is the translation into French?); sequence modeling tasks like named entity recognition (What are the named entity tags in this sentence?) or part-of-speech tagging (What are the part-of-speech tags?); classification problems like sentiment analysis (What is the sentiment?); even multi-sentence joint classification problems like coreference resolution (Who does 'their' refer to?) [119].

6.1 The (20) QA bAbI Tasks

The (20) QA bAbI tasks are a synthetic question and answering dataset from the bAbI project of Facebook AI Research which is organized towards the goal of automatic text understanding and reasoning. It contains 20 tasks, each of them is composed of a set of sentences[29] (story), a question related to some of that sentences, and followed by an answer, which is mostly a single word (in some tasks, the answer is a set of words). For most cases, only a subset of facts (sentences) is relevant to the given question (called *supporting* facts (SF) which are also included in the training set). All the tasks are noiseless and a human able to read that language can potentially achieve 100% accuracy. Two versions of the data are available, the 1st one has 1K training examples per task and the 2nd has 10K examples per task, while the testing set is 1K examples for both versions [120]. The questions in these tasks are quite hard, they not only require lots of knowledge in natural sciences but also abilities to make inferences,

[29] The sentences are as independent from others as possible.

generalize the concepts, apply the general ideas to the examples and so on. These 20 tasks can be summarized into 11 groups:

1- One, Two, or Three supporting facts (Task 1, 2 and 3): In order to answer the question; one, two, or three supporting sentences have to be used as demonstrated in Figure 6.1.

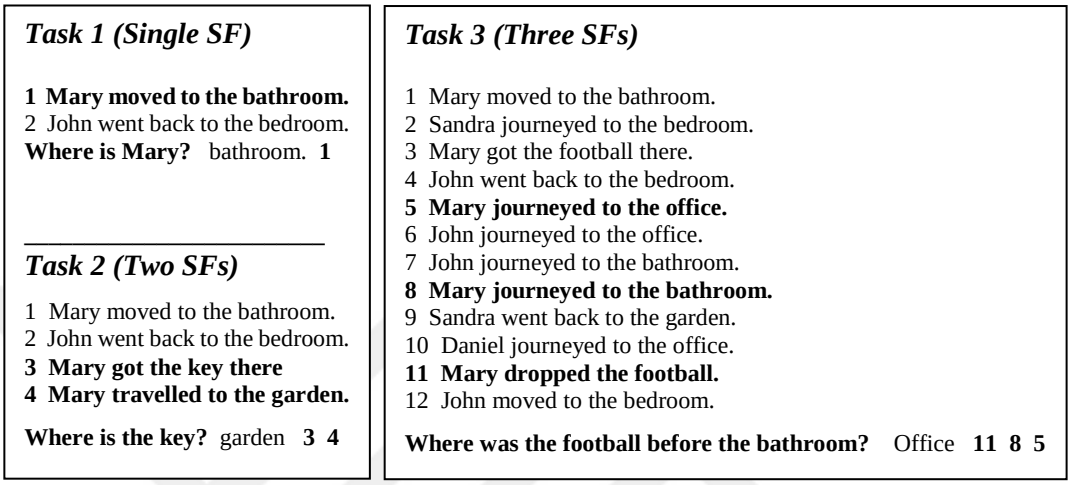


Figure 6.1 One, Two, or Three SFs tasks: Story, question(bold), answer, and indices of the supporting facts (bold).

2- Two or Three argument relations (Task 4 and 5): The ability to differentiate and recognize subjects and objects is necessary to answer the task as illustrated in Figure 6.2.

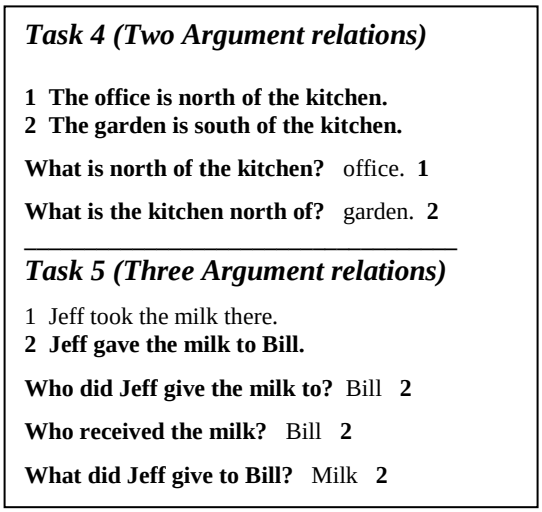


Figure 6.2 Two or Three argument relation tasks: Story, question(bold), answer, and indices of the supporting facts (bold).

Yes/No Questions (Task 6): to test the ability of the model to answer true/false type questions as demonstrated in Figure 6.3.

Task 6 (Yes / No Questions)

1 Daniel went back to the hallway.
2 John got the apple there.
Is Daniel in the hallway? yes. **1**

3 Sandra travelled to the hallway
4 Daniel moved to the bedroom.
Is Daniel in the hallway? no. **4**

Figure 6.3 Yes/No Question Task: Story, question(bold), answer, and indices of the supporting facts (bold).

Counting and Lists/Sets (Task 7 and 8): Task 7 tests the ability of the model to perform the counting operations. Task 8 tests the ability to produce a list of single word answers as demonstrated in Figure **6.4**.

Task 7 (Conting)

1 Mary took the apple there.
2 Sandra travelled to the hallway.
How many objects is Mary carrying? one. **1**

3 Mary travelled to the hallway.
4 Mary got the apple there.
How many objects is Mary carrying? two. **4**

Task 8 (List / Sets)

1 Mary took the apple there.
3 Mary travelled to the hallway.
What is Mary carrying? apple. **1**

3 Sandra travelled to the hallway
4 Mary took the milk there.
What is Mary carrying? apple milk. **1 4**

Figure 6.4 Counting and Lists/Sets: Story, question(bold), answer, and indices of the supporting facts (bold).

Simple Negation and Indefinite Knowledge (Task 9 and 10): Task 9 is similar to Task 6, but with the possibility of negative supporting facts. Task 10 tests if the model can describe the uncertainty as demonstrated in Figure 6.5.

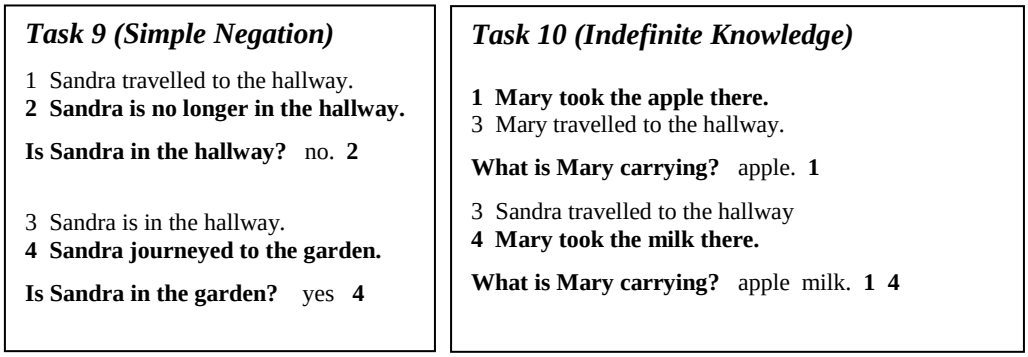


Figure 6.5 Simple negation and Indefinite knowledge: Story, question(bold), answer, and indices of the supporting facts (bold).

Basic Coreference, Conjunctions and Compound Coreference (Task 11, 12 and 13): Task 11 tests the basic type of coreference to detect the single subject. Task 12 tests referring to multiple subjects in a single sentence. Task 13 tests the referring of multiple subjects in multiple sentences as demonstrated in Figure 6.6.

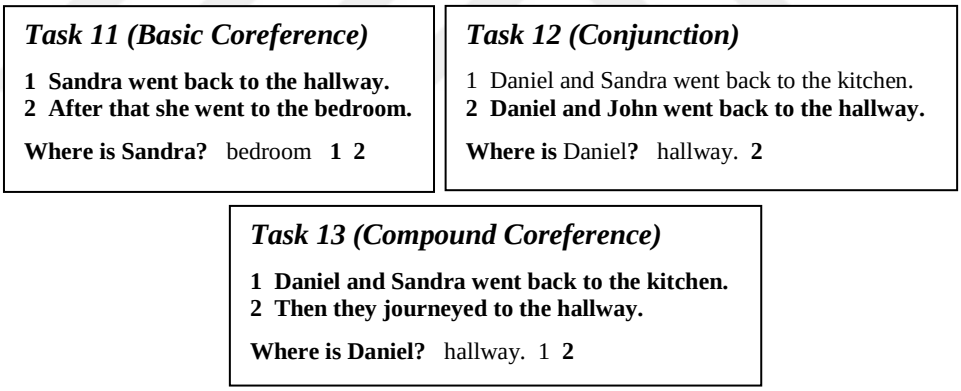


Figure 6.6 Basic Coreference, Conjunctions, and Compound Coreference: Story, question(bold), answer, and indices of the supporting facts (bold).

Time Reasoning: (Task 14): tests the understanding of time expressions within the story as illustrated in Figure 6.7.

.

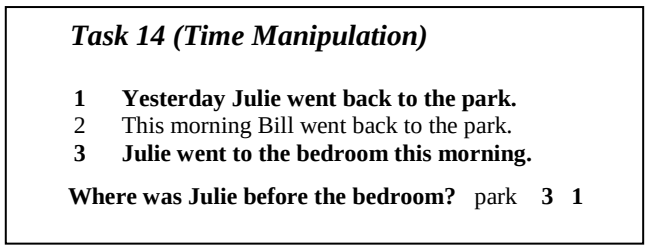


Figure 6.7 Time Reasoning: Story, question(bold), answer, and indices of the supporting facts (bold).

Basic Deduction and Induction (Task 15 and 16): Task 15 tests basic deduction. Task 16 tests basic induction as demonstrated in Figure 6.8.

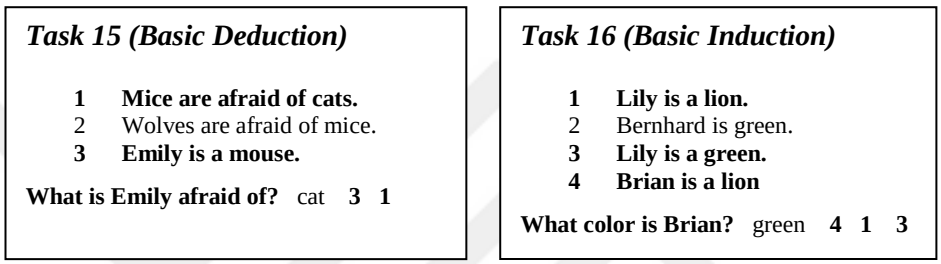


Figure 6.8 Basic Deduction and Induction: Story, question(bold), answer, and indices of the supporting facts (bold).

Positional and Size Reasoning (Task 17 and 18): Task 17 tests the spatial reasoning about the relative positions. Task 18 tests the understanding of the relative size of objects as illustrated in Figure 6.9.

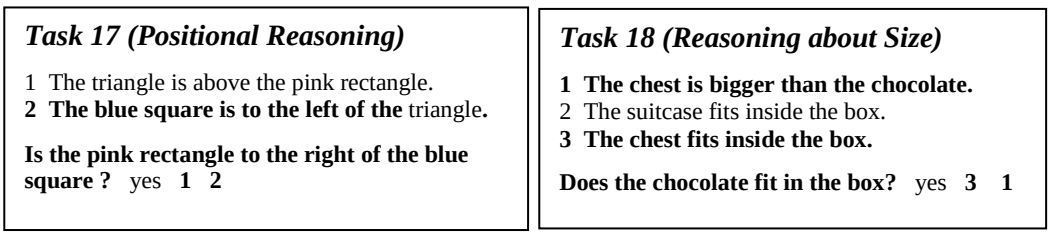


Figure 6.9 Positional and Size Reasoning: Story, question(bold), answer, and indices of the supporting facts (bold).

Path Finding (Task 19): To test the ability to find the path between locations as demonstrated in Figure 6.10.

Task 19 (Path Finding)

1 The kitchen is west of the garden.
2 The garden is south of the office.
3 The office is south of the bedroom.

How do you go from the garden to the bedroom? n n **2 3**

Figure 6.10 Path Finding: Story, question(bold), answer, and indices of the supporting facts (bold).

Agents Motivations (Task 20): tests *why* the agent performs an action and *why* the action has been done.

Task 20 (Reasoning about Motivations)

1 Summit is bored.

Where will Summit go? garden **1**

2 Yann is hungry.

Where will Yann go? kitchen **2**

Figure 6.11 Agents Motivations: Story, question(bold), answer, and indices of the supporting facts (bold).

6.2 Related Work

The 20 QA bAbI tasks have been studied within the context of the Memory Network (MemNN) model [96,120], which consists of four learnable modules: the I-module encodes the input into feature representation, the G-module updates relevant memory slots, the O-module performs inferences to compute output features given the input representation and the current memory, and finally the R-module decodes the output feature-based representation to the final response. Since the proposal of the basic MemNN [96] model, the Adaptive/Nonlinear MemNN [120], Dynamic Memory Networks (DMN)[30] [119], and End-to-End Memory Networks (MemN2N) [123] models have been developed by varying certain parts of these modules. The difference for MemN2N is that the indices of supporting facts are no longer provided in the dataset (weakly supervised) while in the other models the indices of supporting facts are used in training process (strongly supervised). Hence, the MemN2N model must

[30] In DMN Global Vectors for Word Representation (GloVe) [122] has been used instead of the random numbers for word representation in the other methods.

deduce for itself at training and test time which sentences are relevant and which are not; making MemN2N harder and more generally applicable in realistic settings. MemN2N can also be considered as an extension of RNNsearch [124] to the case where multiple computational steps (hops) are performed per output symbol [123]. Within the context of Deep Learning (DL), Neural Machine Translation (NMT) and Neural Turing Machine (NTM) have been proposed to solve the 20 bAbI tasks [125].

Table 6.1 The 20 QA bAbI tasks results: *Facebook* team work on MemNN [120] and MemN2N [123], MitaMind Lab works on DMN [119], *Microsoft* team work on reasoning in vector space (TPR model) [121], *IBM* team work on NMT and NTM [125], and Noah's Ark Lab, *Huawei* Technologies team work on NR [126]. LSTM results are obtained from [120] using the LSTM created in [13].

Task No	Strongly supervised					Weakly supervised			
	MemNN	DMN	NMT	NTM	TPR	MemN2N	NR	NMT	LSTM
1	100%	100%	100%	100%	100%	100%		98.2%	50%
2	100%	98.2%	99.6%	100%	100%	91.7%		41.3%	20%
3	100%	95.2%	99.5%	100%	100%	59.7%		33.4%	20%
4	100%	100%	97.5%	100%	100%	97.2%		97.8%	61%
5	98%	99.3%	90.6%	73.7%	99.8%	86.9%		90.3%	70%
6	100%	100%	99.8%	100%	100%	92.4%		84.6%	48%
7	85%	96.9%	96.6%	100%	100%	82.7%		82.4%	49%
8	91%	96.5%	92.7%	98%	100%	90%		70.8%	45%
9	100%	100%	99.7%	100%	100%	86.8%		89.3%	64%
10	98%	97.5%	96.8%	85.9%	100%	84.9%		73.5%	44%
11	100%	99.9%	100%	100%	100%	99.1%		99.8%	62%
12	100%	100%	100%	100%	100%	99.8%		99.4%	74%
13	100%	99.8%	100%	100%	100%	99.6%		99.7%	94%
14	99%	100%	97.5%	100%	100%	99.3%		44.4%	27%
15	100%	100%	92.7%	100%	100%	100%		42.9%	21%
16	100%	99.4%	88.1%	100%	99.5%	98.7%		42.7%	23%
17	65%	59.6%	58%	61.2%	100%	49%	66.4%	64.6%	51%
18	95%	95.3%	91.8%	93%	100%	88.9%		90.9%	52%
19	36%	34.5%	29.7%	100%	100%	17.2%	17.3%	9.3%	8%
20	100%	100%	93.3%	100%	100%	100%		91.6%	91%
Mean accuracy	93.3%	93.6%	91.2%	95.6%	99.97%	86.1%		72.3%	48.7%

Neural Reasoner (NR) is another approach using the deep architecture, which is a framework for neural network-based reasoning over natural language sentences. NR has two essential specifications: **1**. An interaction-pooling mechanism which allows NR to examine multiple facts, and **2**. a deep architecture, allowing it to deal with the complicated logical relations in reasoning tasks [126], in which only the most difficult tasks (task 17 Positional Reasoning and task 19 Path Finding) from the 20 tasks have been tested and gave superior results for both tasks in weakly supervised.

Finally, in [121] the authors propose vector-space model inspired by Tensor Product Representation (TPR) [92], which achieves the best results for all the 20 bAbI tasks, but with using the *language principles* in the preprocessing of the QA bAbI dataset. The human intervention, e.g., using the *language principles* is not allowed in the 20 QA bAbI tasks from Facebook [120], because the target of the Facebook research team is to produce a language-independent model.

Table 6.1 lists the best results obtained from different models that were used to solve question answering bAbI tasks and indicates that using supporting facts (Strongly supervised) increases the accuracy due to the extra information provided by the task. The most difficult tasks for all models are task 17 (Positional Reasoning) and 19 (Path Finding) because they require a general search algorithm to be built into the inference procedure [120], only TPR model in [121] could solve both tasks due to using external resources, not in the training data (the language principles that has been used in the TPR model).

6.3 Training Methods

The 20 QA bAbI Tasks dataset can be trained in three ways:

1- Using only supporting facts (sentences) and removing all distractor facts.
2- Using all facts (all story) with the indices of supporting facts, this method is called (strongly supervised).
3- Using only all facts (all story) without the indices of supporting facts this method is called (weakly supervised).

In this dissertation, we adopt the first way to check whether ReCA can solve these tasks or not[31].

Figure 6.12 shows the original story of task 3 as an example while Figure 6.13 presents task3 using only supporting facts and inserting the question at the end of the story.

[31] Because, if ReCA could not solve bAbI tasks using only Supporting facts, a fortiori it can not solve them using strongly or weakly supervised

Figure 6.14 demonstrated how to convert the story and question to a matrix using random numbers to represent all words.

Task 3 (Original)

1 Mary moved to the bathroom.
2 Sandra journeyed to the bedroom.
3 Mary got the football there.
4 John went back to the bedroom.
5 Mary journeyed to the office.
6 John journeyed to the office.
7 John journeyed to the bathroom.
8 Mary journeyed to the bathroom.
9 Sandra went back to the garden.
10 Daniel journeyed to the office.
11 Mary dropped the football.
12 John moved to the bedroom.

Where was the football before the bathroom? Office **11 8 5**

Figure 6.12 The original story of task3 where supporting facts (bold), question, answer, and indices of supporting facts (bold numbers).

Task 3 (Only Supporting Facts)

Mary journeyed to the office.
Mary journeyed to the bathroom.
Mary dropped the football.

Where was the football before the bathroom? Office

Figure 6.13 Using only supporting facts for task 3 in Figure 6.12: dataset will be used for training and testing.

	Input							T
Office	2	3	4	5	6	0	0	1
	2	3	4	5	7	0	0	2
The *output*	2	8	5	9	0	0	0	3
label is **6**	10	11	5	9	12	5	7	4

Figure 6.14 Input and output of task3 (Only supporting facts): Converting the story and question in Figure 6.13 to a matrix after representing each word by a number. Thus, the input matrix is 4 × 7 (4 time steps, i.e., T=4) and output is labeled by 6 (means office). The number 1 did not use because it is reserved to represent the space between sentences.

Then, binarizing the decimal numbers in the input matrix and the output number of Figure 6.14 using one hot encoding as in Section 5.2. The binarized matrix is the input sequence **u** that will be used in ReCA as in previous chapters. The target is that ReCA can capture the pattern of the input sequence to predict correctly the output, thus our model is language-independent, i.e., the model can be used for any language.

6.4 Results

After preparing the dataset as in previous section, ReCA is implemented as in Chapter 3, where I is the number of CA evolution iterations in the reservoir and Rule No is the number of ECA rule has been used to achieve best results.

ReCA could solve 15 tasks with 100% accuracy and 2 tasks above 90%, whilst 3 tasks less than 90%. most of tasks achieve best results with rule 90 only 2 tasks use rules 60 and 150. The options ALL and LAST are almost the same as listed in Table 6.2.

Table 6.2 ReCA accuracy for all 20 QA bAbI tasks using ALL and LAST options, where I is the number of CA iterations in the reservoir.

Task	Method	Rule No	I	Accuracy
1	ALL/LAST	90	1	100%
2	ALL/LAST	90	2	100%
3	ALL/LAST	90	1	100%
4	ALL/LAST	90	1	100%
5	ALL/LAST	90	56	100%
6	ALL/LAST	90	12	100%
7	ALL/LAST	60/150	17/4	77%/76%
8	ALL/LAST	90	4	91%/90%
9	ALL/LAST	90	8	100%
10	ALL/LAST	150	39	96%
11	ALL/LAST	90	2	100%
12	ALL/LAST	90	2	100%
13	ALL/LAST	90	3	100%
14	ALL/LAST	90	1/6	100%/90%
15	ALL/LAST	90	2	100%
16	ALL/LAST	90	2	100%
17	ALL/LAST	60/90	128/99	43%/44%
18	ALL/LAST	90	22/17	100%
19	ALL/LAST	90	56/84	38%/39%
20	ALL/LAST	90	17	100%

For comparison, there is only the IBM research team in [125] have used only SFs as our work. The results of their two methods; Neural Machine Translator and Neural Turing Machine are listed in Table 6.3, which shows that ReCA has very good results

where it outperforms in 4 tasks (4, 5, 14, and 18) and falls behind also in 4 tasks (8, 14, 17, and 19). The low number for ReCA in mean accuracy due to very poor accuracy in tasks 17 and 19 compared with NMT and NTM which affected dramatically on the mean accuracy.

Table 6.3 Comparison between ReCA, NMT, and NTM using only supporting facts: the results of NMT and NTM from [125].

Task No	RECA		NMT	NTM
	ALL	LAST		
1	100%	100%	100%	100%
2	100%	100%	100%	100%
3	100%	100%	100%	100%
4	100%	100%	99.1%	100%
5	100%	100%	99.3%	79.2%
6	100%	100%	100%	100%
7	77%	76%	68.5%	100%
8	91%	90%	99%	100%
9	100%	100%	100%	100%
10	96%	96%	98.9%	94.6%
11	100%	100%	100%	100%
12	100%	100%	100%	100%
13	100%	100%	100%	100%
14	100%	90%	99.8%	100%
15	100%	100%	100%	100%
16	100%	100%	100%	100%
17	43%	44%	64.2%	69.3%
18	100%	100%	97.8%	93%
19	38%	39%	80.7%	100%
20	100%	100%	100%	100%
Mean Accuracy	92.25%	91.75%	95.37%	96.81%

6.5 Discussions

After the success of ReCA in (Real and Artificial), (Binary and Nonbinary), and (Static and Dynamic) tasks in previous chapters. ReCA has been tested in this chapter by the 20 QA bAbI tasks from Facebook. bAbI tasks are very hard and complex natural language processing (NLP) and require an understanding of the meaning of a text and the ability to reason over relevant facts.

ReCA could solve most of bAbI tasks 15 out of 20 has 100% accuracy and 2 tasks above 90%, whilst 3 tasks less than 90%. Most of tasks achieve best results with rule 90 only 2 tasks using rules 60 and 150. Hence, all the successful rules are linear (additive) to maximize one-to-one correspondence between the input sequence and the

reservoir activity due to the sequence, i.e., reducing the unwanted interference. The options ALL and LAST are almost the same because the sequence is not long in the most tasks. The harder tasks need a greater number of iterations to achieve high accuracy, i.e., to improve computational power [7, 85]. The most difficult tasks for ReCA are 17 (Positional Reasoning) and 19 (Path Finding) with a large difference compared with NMT and NTM which made the mean accuracy of ReCA is the lowest. The tasks 17 and 19 are the hardest tasks for the most models as explained in Section 6.2. The task 7 (Counting) can also be considered as a hard task for ReCA (accuracy 77%) because the answer is not a word in the story also needs mathematical operations (adding and subtracting).

CHAPTER 7

CONCLUSION AND FUTURE WORK

Reservoir computing based on cellular automata ReCA constructs a novel bridge between automata computational theory and recurrent neural architectures. In this thesis, ReCA has been developed to solve different types of tasks. Several methods have been proposed to extract the features from the cellular automata reservoir. In most tasks, ReCA results outperform the state-of-the-art results.

Concerning the model complexity, a sparsely connected network with simple binary units like elementary cellular automata in ReCA could perform the computational requirements of the reservoir in order to solve hard sequence tasks that have long term dependencies. Thus, ReCA can be considered to operate around the lower bound of complexity.

Sequence learning is an essential capability for a wide collection of intelligence tasks such as language, continuous vision, symbolic manipulation in a knowledge base, etc. Therefore, ReCA has been tested using pathological synthetic tasks of sequence learning that are widely used in RNNs field. ReCA achieves zero error in all pathological tasks; using only the CA evolution states, at last time step, as a feature vector to predict the output (LAST method). CHAPTER 2 shows that the results of LAST outperform the state-of-the-art results in pathological synthetic tasks.

The CA evolution states at all time steps (ALL method) can be used instead of using only the last time step states. Thus, the increasing of provided information using ALL method allows ReCA to solve longer tasks (large T) and using lower training examples but, of course, with large complexity (dimension of feature vector). To overcome this disadvantage, few k states can be used (Each option) rather than using all states in ALL method, exploiting the large information of ALL and low complexity of Each. Further reduction of complexity can be obtained using only on side of the CA evolution states (Half option) or reducing the number of used columns in the matrix of CA evolution states $\mathbf{CA}_{out}$ by selecting small value of expansion ratio *f,* i.e., reducing the dimension

of the zero buffers. CHAPTER 3 shows that the dimension of features (model complexity) is reduced in some tasks by up to 98% for training and 94% for testing. This large reduction is due to the capability of using multiple options together (Each, Half and *f*). Therefore, these results lead us to argue that *CA* evolution states in the reservoir have very rich dynamics, which is why it could reduce the complexity to these large values and why they can also be used to solve more complicated tasks.

Another insertion function (Overwrite) is used instead of XOR in the CA reservoir, which increases the model accuracy with decreasing the required training examples in generalized 5-bit task.

In Chapter 4, The distributed representation of CA in recurrent architecture (ReCA) could solve the 5-bit tasks with minimum complexity, using only *two* training examples which is the lowest number of training examples for any model. Comparing between different architectures and data representations; CA distributed representation in recurrent architecture (ReCA) outperforms the local representation in recurrent architecture (stack reservoir), then echo state networks and feedforward architecture using local or distributed representation. Extracted features from the reservoir, using the natural diffusion of CA states in the reservoir with rule 165 (additive rule) for 5-bit task, and rules 42 and 170 (shift rules) for generalized 5-bit task offers the state-of-the-art results in terms of feature vector dimension and the required training examples. Another extension is obtained by combining the reservoir CA states using XOR, Binary, or Gray operator to produce a single feature vector to reduce the feature space. This method gives promising results, however using the natural diffusion of CA states still outperform a little bit.

After testing ReCA using the pathological synthetic tasks, that are a binary and time-dependent dataset, ReCA has been examined using other types of tasks in order to extend the ReCA applications. Starting by a simple signal classification task then the Japanese vowels task, which is real multidimensional dynamic pattern recognition, the previous both tasks are time-dependent tasks. Finally, tough ReCA is designed for sequence learning; it will be tested using static IRIS dataset. All those tasks are nonbinary dataset; therefore, one hot encoding has been used to binarize the dataset. In signal classification task, ReCA performance was perfect and outperforms the

related work that uses ReCA with a different approach in [51], also ReCA results have the smallest NMSE comparing to the state-of-the-art results. Whilst, ReCA could solve the Japanese vowels task with competitive results 3.5% test error compared with 1.1% to 2.7% test error in the-state-of-the-art machine learning methods. For IRIS dataset, the recurrent architecture could achieve a zero test error while feedforward could not solve this task with zero error due to the high order statistics and distributedness which are provided by the recurrent architecture. The ReCA performance in recurrent architecture outperforms the related work that uses ReCA with different approach reported in [51], where the test error was 3.8% with very high effort compared with ReCA which has a zero test error with a very low effort.

Finally, ReCA has been tested using the 20 QA bAbI tasks from Facebook; These tasks are very hard and require an understanding of the meaning of a text and the ability to reason over relevant facts. Using only supporting facts, ReCA could solve most of bAbI tasks 15 out of 20 has 100% accuracy and 2 tasks above 90%, whilst 3 tasks less than 90%. The options ALL and LAST are almost the same because the sequence is not long in these tasks. The ReCA results are very close to the state-of-the-art results, that provided from Neural Machine Translation and Neural Turing Machine models from IBM research group.

In addition, the usage of cellular automata in the reservoir computing paradigm greatly simplifies the architecture, makes the computation more transparent for analysis, and provide enough computation for large domain of tasks. Furthermore, the reservoir in ReCA can be implemented using ordinary logic gates or Field programmable gate arrays FPGAs, resulting in reducing the complexity in space, time and power consumption.

7.1 Future Work

ReCA framework is very novel, hence there is a lot of research should be done. These are some points to enrich the ReCA research field:

- Using ECA with memory (ECAM) in the reservoir to study the effect of large distributed representation provided by ECAM rules, and even using 2-dimensional CA, e.g., the game of life which has a universal computation.
- Using the rest 256 ECA rules that have not been used in this dissertation to explore their specifications.
- Preprocessing the input before the reservoir as in [8] where it provided promising results.
- Other training methods can be used in the read-out stage rather than linear regression, e.g. logistic regression, support vector machines SVMs or even feedforward neural networks in order to augment the model computational power.
- Overwrite insertion function should be studied extensively due to its promising results in Chapter 4.
- Using hybrid ReCA (multiple rules in the reservoir) as in [49, 51] or deep ReCA (cascade models) as in [50] to solve more complicated tasks.
- In QA tasks, adding Attention mechanisms in order to allow ReCA to focus on a specific part of the input (supporting facts) leading to use ReCA in weakly supervised.

REFERENCES

[1] Samarasinghe, S., Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition. CRC Press, 2016.

[2] Poznyak, T. I., Oria, I. C., & Poznyak, A. S., Ozonation and Biodegradation in Environmental Engineering, Dynamic Neural Network Approach. (Chapter 3: Background on dynamic neural networks), Elsevier Inc., 58-74, 2019.

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

[4] Jaeger, H., The 'Echo State' Approach to Analyzing and Training Recurrent Neural Networks—with an Erratum Note. GMD Technical Report, 148:34, Bonn, Germany: German National Research Center for Information Technology, 2001.

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

[6] Steil, J. J., Backpropagation-decorrelation: online recurrent learning with O(N) complexity. In IEEE International Joint Conference on Neural Networks, 2, 843-848, 2004.

[7] Yilmaz, O., Machine Learning using Cellular Automata based Feature Expansion and Reservoir Computing. Journal of Cellular Automata, 10(5-6), 435-472, 2015.

[8] Margem, M., & Yilmaz, O., How much computation and distributedness is needed in sequence learning tasks?" In Artificial General Intelligence, AGI-16, Lecture Notes in Computer Science, Springer, 9782, 274-283, 2016.

[9] Margem, M., & Gedik, O. S., Reservoir Computing Based on Cellular Automata (ReCA) in Sequence Learning. Journal of Cellular Automata. 14(1-2), 153-170, 2019.

[10] Funahashi, K., & Nakamura, Y., Approximation of dynamical systems by continuous time recurrent neural networks. Neural networks, 6(6), 801-806, 1993.

[11] Siegelmann, H. T., & Sontag, E. D., On the computational power of neural nets. Journal of computer and system sciences, 50(1), 132-150, 1995.

[12] Doya, K., Bifurcations in the learning of recurrent neural networks. In IEEE International Symposium on Circuits and Systems, 2777–2780, 1992.

[13] Hochreiter, S., & Schmidhuber, j., Long short-term memory" Neural computation, 9(8), 1735-1780, 1997.

[14] Al Rodan, A. A., Architectural design of Echo state network. School of Computer Science, College of Engineering and Physical Sciences, The University of Birmingham, Ph.D. Thesis, 2012.

[15] Atiya, A. F., & Parlos, A. G., New results on recurrent network training: Unifying the algorithms and accelerating convergence. IEEE Transactions on Neural Networks, 11, 697-709, 2000.

[16] Tino, P., & Dorffner, G., Predicting the future of discrete sequences from fractal representations of the past. Machine Learning. 45(2), 187-218, 2001.

[17] Ishii, K., van der Zant, T., Becanovic, V., & Ploger, P., Identification of motion with echo state network. In Proceedings of the Oceans 2004 MTS/IEEE -Techno-Ocean Conference, 3 1205-1210, 2004.

[18] Bush, K., & Anderson, C., Modeling reward functions for incomplete state representations via echo state networks. In Proceedings of the International Joint Conference on Neural Networks, Montreal, Quebec, 2005.

[19] Deng, Z., & Y., Zhang., Collective behavior of a small-world recurrent neural system with scale-free distribution. IEEE Transactions on Neural Networks, 18(5), 1364-1375, 2007.

[20] Jones, B., Stekel, D., Rowe, J., & Fernando, C., Is there a liquid state machine in the bacterium escherichia coli?. In Proceedings of the 2007 IEEE Symposium on Artificial Life (CI-Alife), 18-191., 2007.

[21] Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F., Training recurrent networks by evolino. Neural Computation, 19, 757-779, 2007.

[22] Rad, A. A., Jalili, M., & Hasler, M., Reservoir optimization in recurrent neural networks using kronecker kernels. In IEEE International Symposium on Circuits and Systems, 868-871, IEEE, 2008.

[23] Dockendorf, K. P., Park, I., Ping, H., Príncipe, J. C., & DeMarse, T. B., Liquid state machines and cultured cortical networks: The separation property. Biosystems, 95(2), 90-97, 2009.

[24] Jaeger, H., A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the ‘echo state network’ approach. Technical report GMD report 159, German National Research Center for Information Technology, 2002.

[25] Jaeger, H., & Hass, H., Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. Science, 304, 78-80, 2004.

[26] Skowronski, M. D., & Harris, J. G., Minimum mean squared error time series classification using an echo state network prediction model. In IEEE International Symposium on Circuits and Systems, Island of Kos, Greece, 3153-3156, 2006.

[27] Tong, M. H., Bicket, A. D., Christiansen, E. M., & Cottrell, G. W., Learning grammatical structure with echo state network. Neural Networks, 20, 424-432, 2007.

[28] Soh, H., & Demiris, Y., Iterative temporal learning and prediction with the sparse online echo state gaussian process. In International Joint Conference on Neural Networks (IJCNN), 1-8, IEEE, 2012.

[29] Jalalvand, A., Van Wallendael, G., & Van de Walle, R., Real-time reservoir computing network-based systems for detection tasks on visual contents. In 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), 146-151, IEEE, 2015.

[30] Maass, W., Natschläger, T., & Markram, H., Fading memory and kernel properties of generic cortical microcircuit models. Journal of Physiology-Paris 98, 315-330, 2004.

[31] Yildiz, I. B., Jaeger, H., & Kiebel, S. J., Re-visiting the echo state property. Neural networks, 35, 1-9, 2012.

[32] Lukoševičius, M., & Jaeger, H., Reservoir computing approaches to recurrent neural network training. Computer Science Review, 3(3), 127-149, 2009.

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

[34] Legenstein, R., & Maass, W., Edge of chaos and prediction of computational performance for neural circuit models. Neural Networks. 20(3), 323-334, 2007.

[35] Lukoševičius, M., A Practical Guide to Applying Echo State Networks. In Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science, 659-686, Springer, 2012,

[36] Adamatzky, A., Computing in nonlinear media and automata collectives. CRC Press, 2001.

[37] Fernando, C., & Sojakka, S., Pattern recognition in a bucket. In European Conference on Artificial Life (ECAL 2003), 588-597, Springer, 2003.

[38] Dai, X., Genetic Regulatory Systems Modeled by Recurrent Neural Network. In Proceedings, Part II, Advances in Neural Networks: International Symposium on Neural Networks (ISNN 2004), 519-524, Springer, 2004.

[39] Jones, B., Stekel, D., Rowe, J., & Fernando, C., Is There a Liquid State Machine in the Bacterium Escherichia coli?. In Proceedings of the IEEE Symposium on Artificial Life 2007 (ALIFE'07), 187-191, IEEE, 2007.

[40] Paquot Y., Duport, F., Smerieri, A., Dambre, J., Schrauwen, B., Haelterman, M., & Massar, S., Optoelectronic Reservoir Computing. Scientific Reports in Nature, 2, 287, 2012. doi:10.1038/srep00287

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

[42] Ortín, S., Soriano, M. C., Pesquera, L., Brunner, D., San-Martín, D., Fischer, I., Mirasso C. R., & Gutiérrez, J. M., A unified framework for reservoir computing and extreme learning machines based on a single time-delayed neuron. Scientific reports, 5, 14945, 2015.

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

[44] Dale, M., Miller, J. F., Stepney, S., & Trefzer, M. A., Evolving Carbon Nanotube Reservoir Computers. In Proceedings of International Conference on Unconventional Computation and Natural Computation (UCNC 2016), 49-61, Springer International Publishing, 2016. doi:10.1007/978-3-319-41312-9_5.

[45] Dale, M., Miller, J. F., & Stepney, S., Reservoir Computing as a Model for in-Materio Computing. Advances in Unconventional Computing: Volume 1: Theory (Adamatzky, A., ed.), 533-571, Springer International Publishing, 2017.

[46] Goudarzi, A., Lakin, M.R., & Stefanovic, D., DNA reservoir computing: a novel molecular computing approach. In International Workshop on DNA Based Computers, 76-89, Springer, 2013.

[47] Yamane, T., Katayama, Y., Nakane, R., Tanaka, G., & Nakano, D., Wave based reservoir computing by synchronization of coupled oscillators. In International Conference on Neural Information Processing, 198-205, Springer, 2015.

[48] Coulombe, J.C., York, M.C., & Sylvestre, J., Computing with networks of nonlinear mechanical oscillators. PloS one, 12(6), e0178663. 2017.

[49] Nichele, S., & Gundersen, M. S., Reservoir Computing Using Non-Uniform Binary Cellular Automata. Complex Systems, 26(3), 225-245, Complex Systems Publications Inc., 2017.

[50] Nichele, S., & Molund, A., Deep learning with cellular automaton-based reservoir computing. Complex Systems, 26(4), 319-339, Complex Systems Publications Inc., 2017.

[51] McDonald, N., Reservoir Computing & Extreme Learning Machines using Pairs of Cellular Automata Rules. In International Joint Conference on Neural Networks (IJCNN), USA, 88, 2429-2436, 2017.

[52] Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., & Hirose, A., Recent advances in physical reservoir computing: A review. Neural Networks, 2019. doi: 10.1016//j.neunet.2019.03.005.

[53] Hadaeghi, F., He, X., Jaeger, H., Unconventional Information Processing Systems, Novel Hardware: A Tour D'Horizon. IRC-Library, Information Resource Center der Jacobs University Bremen, 2017.

[54] von Neumann, J., The General and Logical Theory of Automata. In L.A. Jeffress (ed.), Cerebral Mechanisms in Behavior: The Hixon Symposium, 1-31, New York, John Wiley, 1951.

[55] von Neumann, J., Theory of Self-Reproducing Automata. Urbana: University of Illinois Press (ed. A.W. Burks), 1966.

[56] Burks, A. W., (ed.), Essays on Cellular Automata. University of Illinois Press, 1970.

[57] Frisch, U., Hasslacher, B., & Pomeau. Y., Lattice-gas automata for the Navier-Stokes equation. Physical Review Letters, 56(14), 1505-1508, 1986.

[58] Toffoli, T., & Margolus, N., Cellular automata machines: a new environment for modeling. MIT Press, 1987.

[59] L., Lam, (ed.), Nonlinear Physics for Beginners: Fractals, chaos, solitons, pattern formation, cellular automata and complex systems. World Scientific, 1998.

[60] Shackleford, B., Tanaka, M., Carter, R. J., & Snider, G. FPGA implementation of neighborhood-of-four cellular automata random number generators. In Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays, 106-112, ACM, 2002.

[61] Gobron, S., Devillard, F., & Heit, B. Retina simulation using cellular automata and GPU programming. Machine Vision and Applications, 18(6), 331-342, 2007.

[62] Powley, E. J., Global properties of cellular automata. PhD Thesis, University of York, Department of Computer Science, 2009.

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

[64] Moore, E.F., Machine models of self-reproduction. In Proceedings of symposia in applied mathematics, 14, 17-33, American mathematical society, New York, 1962.

[65] Gardner, M., The fantastic combinations of John Conway’s new solitaire game of life. Sci. Am., 223, 120–123, 1970.

[66] Jump, J. R., & Kirtane, J. S., On the interconnection structure of cellular automata networks. Information and Control, 24(1), 74-91, 1974.

[67] Dyer, C., One-way bounded cellular automata. Information and Control 44(3), 261-281, 1980.

[68] Boccara, N., & Fuks, H., Cellular automaton rules conserving the number of active sites. Journal of Physics A: Mathematical and General, 31(28), 6007, 1998.

[69] Wolfram, S., Statistical mechanics of cellular automata. Reviews of modern physics, 55(3), 601-644, 1983.

[70] Li, W., & Packard, N., The Structure of the Elementary Cellular Automata Rule Space. Complex Systems, 4(3), 281-297, 1990.

[71] Martínez, G. J., A Note on Elementary Cellular Automata Classification. Journal of Cellular Automata, 8(3-4), 233-259, 2013.

[72] Salman, K., Analysis of elementary cellular automata boundary conditions. International Journal of Computer Science & Information Technology, 5(4), 35, 2013.

[73] Bhattacharjee, K., Naskar, N., Roy, S., & Das, S. A survey of cellular automata: types, dynamics, non-uniformity and applications. Natural Computing, 1-29, 2018.

[74] Martínez, G., Seck-Tuoh-Mora J., & Zenil H., Computation and Universality: Class IV versus Class III Cellular Automata. Journal of Cellular Automata, 7(5-6), 393-430, 2013.

[75] Martínez, G. J., Seck-Tuoh-Mora, J. C., & Zenil, H. Wolfram's classification and computation in cellular automata classes III and IV. Irreducibility and Computational Equivalence, Zenil, H. (ed.), Chapter 17, 237-259, Springer, 2013.

[76] Cook, M., Universality in elementary cellular automata. Complex Systems, 15(1), 1-40, 2004.

[77] Langton, C. G., Studying Artificial Life with Cellular Automata. Physica D: Nonlinear Phenomena, 22(1-3), 120-149, 1986.

[78] Toffoli, T., & Margolus, N., Cellular Automata Machines. The MIT Press, 1987.

[79] Martínez, G. J., Adamatzky, A., & Alonso-Sanz, R., "Designing complex dynamics in cellular automata with memory. International Journal of Bifurcation and Chaos in Applied Sciences and Engineering, 23(10), 1330035, 2013.

[80] Alonso-Sanz, R., & Martin, M., Elementary cellular automata with elementary memory rules in cells: The case of linear rules. Journal of Cellular Automata, 1(1), 71-87, 2006.

[81] Yilmaz, O., Reservoir computing using cellular automata. arXiv preprint arXiv:1410.0162v1, 2014.

[82] Yilmaz, O., Symbolic Computation using Cellular Automata based Hyperdimensional Computing. Neural Computation, 27(12), 2661-2692, 2015.

[83] Yilmaz, O., Analogy Making and Logical Inference on Images using Cellular Automata based Hyperdimensional Computing. In NIPS, Workshop on Cognitive Computation, 2015

[84] Morán, A., Frasser, C. F., & Rosselló, J. L., Reservoir Computing Hardware with Cellular Automata. arXiv preprint arXiv:1806.04932v2, 2018.

[85] Margem, M., & Gedik, O. S., Feed-forward vs. Recurrent Architecture and Local vs. Cellular Automata Distributed Representation Based Reservoir Computing in Sequence Memory Learning. under review in Artificial Intelligence Review (AIRE), 2019.

[86] Nakayama, A., Yamamoto, T., Morita, Y., & Nakamachi, E., Development of multi-layered cellular automata model to predict nerve axonal extension process. In VI International Conference on Computational Bioengineering, 2015.

[87] Zhang, X., Lu, R., Zhang, H., & Xu, C., A New Digital Signature Scheme from Layered Cellular Automata. International Journal of Network Security, 18(3), 544-552, 2016.

[88] Martens, J., Sutskever, I., Learning recurrent neural networks with hessian-free optimization. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), 1033-1040, 2011.

[89] Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J., A Field Guide to Dynamical Recurrent Neural Networks. Chapter Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, IEEE press, 2001.

[90] Jaeger, H., Long Short-Term Memory in Echo State Networks: Details of a Simulation Study. Technical report No. 27, Jacobs University Bremen, 2012.

[91] Pascanu, R., Mikolov, T., & Bengio, Y., On the difficulty of training Recurrent Neural Networks" In the 30th International Conference on Machine Learning, USA, 2013.

[92] Smolensky, P., Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artificial intelligence, 46(1-2), 159-216, 1990.

[93] Deypir, M., Sadreddini, M. H., & Hashemi, S., Towards a variable size sliding window model for frequent itemset mining over data streams. Computers & Industrial Engineering, 63, 161–172, 2012. doi: 10.1016/j.cie.2012.02.008.

[94] Kang, G., & Guo, S., Variable sliding window DTW speech identification algorithm. In 2009 Ninth International Conference on Hybrid Intelligent Systems, 1, 304-307, IEEE, 2009. doi: 10.1109/HIS.2009.66.

[95] Myung, I. J., The importance of complexity in model selection. Journal of mathematical psychology, 44(1), 190-204, 2000.

[96] Weston, J., Chopra, S., & Bordes, A., Memory Networks. International Conference on Learning Representations (ICLR), USA, 2015.

[97] Graves, A., Wayne, G., & Danihelka, I., Neural Turing machines" arXiv preprint arXiv:1410.5401, 2014.

[98] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., et al. Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626), 471, 2016.

[99] Collier, M., & Beel, J., Implementing Neural Turing Machines. Artificial Neural Networks and Machine Learning – ICANN 2018, Lecture Notes in Computer Science, 11141, 94-104, Springer International Publishing, 2018.

[100] Doran, R. W., The Gray Code. Journal of Universal Computer Science, 13(11), 1573-1597, 2007.

[101] Dietterich, T. G., Machine learning for sequential data: A review. Structural, syntactic, and statistical pattern recognition, 15-30. Springer, 2002.

[102] Huang, G. B., Zhu, Q. Y., & Siew, C. K., Extreme learning machine: a new learning scheme of feedforward neural networks. In Proceedings 2004 IEEE International Joint Conference on Neural Networks IJCNN, 2, 985-990, 2004.

[103] Huang, G. B., Wang, D. H., & Lan, Y., Extreme learning machines: a survey. International Journal of Machine Learning and Cybernetics, 2(2), 107-122, 2011.

[104] Wolfram, S., Tables of Cellular Automaton properties 1986. Appendix in Cellular automata and complexity: collected papers, 513-584, Westview Press, 1994.

[105] Kuhn, M., & Johnson, K., Applied Predictive Modeling. Science+Business Media New York, 26, 71-72, Springer, 2013. doi: 10.1007/978-1-4614-6849-3.

[106] Cerda, P., Varoquaux, G., & Kégl, B., Similarity encoding for learning with dirty categorical variables. Machine Learning, 107(8-10), 1477-1494, 2018.

[107] Zhang, H., Feng, X., Li, B., Wang, Y., Cui, K., Liu, F., Dou, W., & Huang, Y., Integrated photonic reservoir computing based on hierarchical time-multiplexing structure. Optics express., 22(25), 31356-31370. 2014.

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

[109] kdd.ics.uci.edu/databases/JapaneseVowels/JapaneseVowels.html, retrieved 1 Oct. 2017.

[110] Geurts, P., Pattern extraction for time series classification. In European Conference on Principles of Data Mining and Knowledge Discovery, 115-127, Springer, 2001.

[111] Barber, D., Dynamic Bayesian networks with deterministic latent tables. In Advances in Neural Information Processing Systems (NIPS). 2003.

[112] Strickert, M., Self-organizing neural networks for sequence processing. Ph.D. thesis, Univ. of Osnabruck, Dpt. of Computer Science. 2004.

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

[114] Jaeger, H., Controlling recurrent neural networks by conceptors. arXiv preprint arXiv:1403.3369, 2017.

[115] Antoniol, G., Rollo, V. F., & Venturi, G., Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories. ACM SIGSOFT software engineering notes, 30(4), 1-5, 2005.

[116] Molund, A., Deep Reservoir Computing Using Cellular Automata. M.Sc. thesis, Norwegian University of Science and Technology, Department of Computer Science, 2017.

[117] Fisher, R., The use of multiple measurements in taxonomic problems. Contributions to Mathematical Statistics, John Wiley, NY, 1950.

[118] archive.ics.uci.edu/ml/datasets/iris, retrieved 20 Nov 2017.

[119] Kumar, A., Ondruska, F., Iyyer, M., Bradbury, J., Gulrajani I., Zhong, V., Paulus, R., & Socher, R., Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. In Proceedings of the 33rd International Conference on Machine Learning in PMLR., 48, 1378-1387, 2016.

[120] Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merrienboer, B., Joulin, A., & Mikolov, T., Towards AI-Complete Question Answering: A set of Prerequisite Toy Tasks. arXiv preprint arXiv: 1502.05698v10, 2015.

[121] Lee, M., He, X., Yih, W. T., Gao, J., Deng, L., & Smolensky, P., Reasoning in Vector Space: An Exploratory Study of Question Answering. arXiv preprint arXiv: 1511.06426v4., 2016.

[122] Pennington, J., Socher, R., & Manning, C., Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 1532-1543, 2014.

[123] Sukhbaatar, S., Weston, J., & Fergus, R., End-to-end memory networks. In Advances in neural information processing systems, 2440-2448, 2015.

[124] Bahdanau, D., Cho, K., & Bengio, Y., Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

[125] Yu, Y., Zhang, W., Hang, C. W., Xiang, B., & Zhou, B., Empirical study on deep learning models for question answering. arXiv preprint arXiv:1510.07526, 2015.

[126] Peng, B., Lu, Z., Li, H., & Wong, K. F., Towards neural network-based reasoning. arXiv preprint arXiv:1508.05508, 2015.

[127] Patel, K., Vala, J., & Pandya, J., Comparison of various classification algorithms on iris datasets using WEKA" International journal of Advance Engineering and Research Development (IJAERD), 1(1). 2014.

[128] WEKA at http://www.cs.waikato.ac.nz/~ml/weka.

CURRICULUM VITAE

PERSONAL INFORMATION

Name Surname **:** Mrwan MARGEM

Date of Birth **:** 21/08/1968

Phone **:** 0090 537 49 35 301

E-mail **:** m_mrwan@yahoo.com

EDUCATION

1982 – 1985

Baccalaureate in sciences, Zawit Al Dehmani higher school, Tripoli / Libya.

1985 – 1990

B.Sc. of Engineering sciences, Electrical & Electronic Dept., Faculty of Engineering, Tripoli University, Tripoli / Libya.

2002 – 2004

M.Sc. of Control Engineering, Henri Poincaré University, Nancy / France.

WORK EXPERIENCE

1991 – 1993

Assistant lecturer in computer laboratory in Electrical & Electronic Dept. at Tripoli University.

1994 – 1996

lecturer in Digital circuits at the Higher Institute of Industrial Technology. Tripoli / Libya

1997 – 2001

lecturer in Electronic circuits at the Higher Institute of Electronic Professions. Tripoli / Libya

2005 – 2007

lecturer in Electronic Instrumentation & Maintenance at the Higher Institute of Electronic Professions. Tripoli / Libya

2008 – 2013

lecturer in Sensors, Data Acquisition & Power Electronics at the College of Electronic Technology-Tripoli.

LANGUAGES

Arabic (Native), English, French.

TECHNICAL SKILLS

Programing Languages: Matlab, Python.

Networking : CISCO Network Academy (CCNA) 1,2,3,4. CCNA certified trainer.

Platforms : Windows 98/2000/XP/Vista/7/8/10.

Tools : Latex, MS Office.

TOPICS OF INTEREST

Control Engineering, Power Electronics, Machine Learning.

PUBLICATIONS

- Margem, M., El-Mezugi, D., & Najmeddin, H., “Comparison between the actual and standard specifications of Op-amp 741 and study their effects on the experiments at the College of Electronic Technology-Tripoli” In the 1st Conference of Technology Education, Ezzawia / Libya, 2013.

- Margem, M., & Yilmaz, O., “How much computation and distributedness is needed in sequence learning tasks?” In Artificial General Intelligence, AGI-16, Lecture Notes in Computer Science, Springer, 9782, 274-283, 2016.

- Margem, M., & Gedik, O. S., “Reservoir Computing Based on Cellular Automata (ReCA) in Sequence Learning. Journal of Cellular Automata. 14(1-2), 153-170, 2019.

- Margem, M., & Gedik, O. S., “Feed-forward vs. Recurrent Architecture and Local vs. Cellular Automata Distributed Representation Based Reservoir Computing in Sequence Memory Learning. under review in Artificial Intelligence Review (AIRE), 2019.