How Much Computation and Distributedness is Needed in Sequence Learning Tasks?

Mrwan Margem[1] and Özgur Yilmaz[1]

Turgut Ozal University, Department of Computer Engineering, Ankara Turkey
mmargem2013@stu.turgutozal.edu.tr ozyilmaz@turgutozal.edu.tr;
~ozguryilmazresearch.net

Abstract. In this paper, we are analyzing how much computation and distributedness of representation is needed to solve sequence-learning tasks which are essential for many artificial intelligence applications. We propose a novel minimal architecture based on cellular automata. The states of the cells are used as the reservoir of activities as in Echo State Networks. The projection of the input onto this reservoir medium provides a systematic way of remembering previous inputs and combining the memory with a continuous stream of inputs. The proposed framework is tested on classical synthetic pathological tasks that are widely used in evaluating recurrent algorithms. We show that the proposed algorithm achieves zero error in all tasks, giving a similar performance with Echo State Networks, but even better in many different aspects. The comparative results in our experiments suggest that, computation of high order attribute statistics and representing them in a distributed manner is essential, but it can be done in a very simple network of cellular automaton with identical binary units. This raises the question of whether real valued neuron units are mandatory for solving complex problems that are distributed over time. Even very sparsely connected binary units with simple computational rules can provide the required computation for intelligent behavior.

1 Introduction

Intelligence requires remembering previously presented perceptual input or a past cognitive state, and making a synthesis to provide an output for effectively interacting with the environment. This capability is essential in any artificial general intelligence system and there are various solutions to this problem. In this study, we are providing a very simple recurrent architecture for solving sequence tasks that poses as a lower bound on complexity. The recurrent formulation is applied for three different representations: Stack, Covariance and Cellular Automata. In Stack representation, there is no computation involved, the input sequence steps are reserved one after another in raw format. Covariance representation computes the pairwise covariance of the input attributes and locally saves those for each step of the sequence input, as in Tensor Products [13]. Being very similar to Covariance representation in terms of complexity of operations,

Cellular Automata holds a distributed representation of high order attribute statistics. Stack representation provides pure memorization, whereas Covariance representation computes useful second order statistics. Cellular Automata representation enables both computation of high order statistics and distributedness. We contrast these three approaches in sequence learning tasks and show that Cellular Automata approach gives superior performance than the other two and equivalent performance with Echo State Networks with significantly less computational demands. Therefore, one of the main contributions of the paper is a novel framework of cellular automata based reservoir computing in a recurrent setting (called *ReCA*), that is capable of short-term memory.

Next, we review reservoir computing and cellular automata, then provide methods and results in the following sections. We discuss the results of our experiments contrasting feedforward vs feedback, memory vs computation, local vs distributed.

1. Reservoir Computing: Many real life problems in artificial intelligence (AI) require the system to remember previous inputs. Recurrent Neural Networks (RNN)s are powerful tools of machine learning with memory. Therefore, they employ very powerful hierarchical computation as well as distributed representation which make them excellent tools for sequence learning tasks. Unfortunately RNNs are difficult to train due to the inherent difficulty of learning optimal representations tailored for long-term dependencies [1,4] and convergence issues [3]. In 2001 an approach to design and train RNNs was proposed independently by Wolfgang Maass and Herbert Jaeger under the names of Liquid State Machines (*LSM*) [9] and Echo State Networks (*ESN*) [5] respectively. These two methods became lately known as Reservoir Computing (RC) approaches [14]. RC model avoids the shortcomings of conventional training methods in RNNs, by setting up RNNs in the following way: **1.** Building the *Reservoir* which is a randomly created RNN and remains unchanged during training. The reservoir is excited by the input signal and maintains in its state a nonlinear transformation of the input history. **2.** The output signal is generated as a linear combination of the neuron's signals from the input-excited reservoir. This linear combination is learned by linear regression, using the teacher signal as a target [8]. A question still remains: how much computation is needed in the reservoir? In Echo State Networks, real valued neurons with nonlinear activation functions are utilized and this still corresponds to a fairly complex neural model. Can we simplify the recurrent architecture further by using identical binary units, and the maximum possible amount of connection sparsity? This corresponds to an elementary cellular automaton array.

2. Cellular Automata: Cellular Automata (CA) are discrete dynamical systems with sparse connections [16]. A cellular automaton is an array of cells evolving synchronously according to an identical interaction rule. The evolution of a cell is dependent on the previous states of a surrounding neighborhood of cells as shown in Fig. 1(b) and thoroughly investigated in [15].

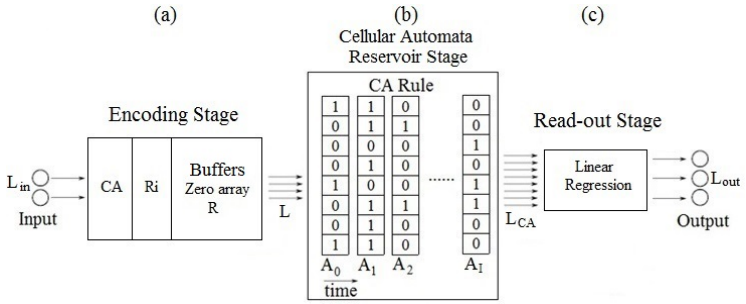

Fig. 1. General framework for Cellular Automata based Reservoir Computing (*ReCA*) with vector lengths for each stage: **a.** Encoding Stage that consists of; ***CA***: Input expansion using *ECA* (Multilayer CA), ***Ri***: Representing each input bit by R_i bits, and **Buffers:** adding zero array R. **b.** Cellular Automata Reservoir Stage; The output of Encoding Stage is projected onto cellular automaton instead of a neural network in Echo State Networks. A_0 is evolved using a pre-specified ECA rule from 1 to I iterations. **c.** Read-out Stage; The feature vector (reservoir output) is trained by Linear regression.

The dimension of cellular automata can have an integer value, thus it is a grid in general, but one or two dimensions are utilized in most of the studies. If the rules change in time (in certain iterations), the configuration is termed as *multi-layer* CA. In our work, we will use exclusively Elementary Cellular Automaton (ECA) which is a one dimensional CA with binary cells (*1* or *0*), evolving according to a uniform (non-changing rule). The ECA rules are classified according to their evolution behavior (Wolfram classes) [15]. Starting from random initial cell values, CA state evolution will show a certain behavior: **Class I** (*Uniform*) CA states evolve to a homogeneous behavior, **Class II** (*Periodic*) CA states evolve periodically, **Class III** (*Chaotic*) CA states evolve chaotically (without any defined pattern) and **Class IV** (*Complex* or *edge of chaos*) can show all these evolution patterns in an unpredictable manner.

2 Cellular Automata in Reservoir Computing: *ReCA*

The introduction of Cellular Automata into reservoir computing framework was proposed in [18] and some applications discussed in [19,17].

The main idea can be summarized as follows: cellular automata provide powerful enough computation and rich enough representation to be used instead of real valued recurrent neural networks. Cellular automaton is a very sparsely connected network with identical and binary units, thus it gives a lower bound on the amount of model complexity for solving hard problems in AI.

In this paper we are using the recurrent formulation of cellular automata reservoir to handle a sequence of inputs. The algorithmic flow of CA based RC (*ReCA*) is shown in Fig. 1. The encoding stage translates the input into the initial states of CA. In cellular automata reservoir stage, the ECA rules are applied for a fixed period of iterations (I), to evolve the CA initial states. The

CA states in the reservoir are concatenated to produce a feature vector that will be used in read-out stage (Linear Regression).

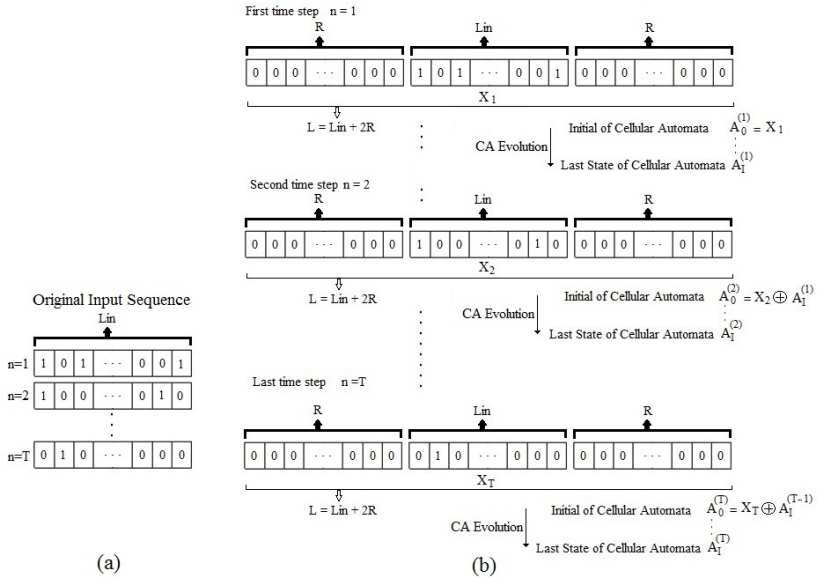

Fig. 2. a. Original input sequence before encoding stage. **b.** Encoding stage and CA Reservoir stage: Adding zero array with length R to obtain the input steps to the reservoir (X_1 to X_T, size L vectors). Then cellular automaton is initialized with the first time step input of the sequence, so $A_0^{(1)} = X_1$, where the subscript 0 denotes to initial state and increases to I (No of CA iterations) and the superscript (1) denotes to the number of time steps (from 1 to T). The CA states $A_i^{(n)}$ are used as the features to estimate the output at time step n ($y(n)$) using linear regression in read-out stage.

2.1 Encoding Stage

In encoding stage, the input is translated into the initial states of cellular automaton. It can be divided into three subroutines as shown in Fig. 1(a)[1]

1. **Utilizing Buffers (Zero Array R):** For handling a sequence of inputs, an array of zeros with length of R are added to both sides of original input, these buffers will hold the activity of the reservoir corresponding to previous time steps. Then, the expanded input to cellular automata reservoir is of fixed length $L = L_{in} + 2R$, as shown in Figs. 1 and 2. In most of the

[1] We should note that, zero buffers are utilized for all experiments but the other two subroutines are applied selectively according to the task.

experiments, R equals to $I \times T$ (I is the number of CA iterations and T is the sequence length of the input), to guarantee that CA states due to all time sequences have been conserved. To relax this constraint and to reduce time/space complexity an expansion ratio $f \in [0, 1]$ has been introduced as follows $R = f * (I \times T)$, thus the size of the reservoir (complexity) decreases with the value of f.

2. **Reducing Interference (R_i):** To improve the accuracy in tasks, the interference between non zero elements in the reservoir should be reduced. Therefore, each bit of the input will be represented by R_i bits[2]. After each time step, the location of the non zeros will rotate right[3] one bit. For more details see [10].
3. **Multilayer Cellular Automata Expansion:** The original binary input can be transformed into another binary vector using nonlinear *ECA* rules to increase the nonlinearity of the model. This stage enables a *Multilayer Cellular Automata* architecture, in which the first layer projects the input into a nonlinear space, and the next layer evolves it further with linear rules in time to expand the feature space. Linearity in the second layer is essential for lossless injection of the input at each time step.

2.2 Cellular Automata Reservoir Stage

After the input data is encoded as the initial states of a cellular automaton, it is passed on a CA reservoir (instead of an ESN as in [5]) for computation as shown in Fig. 1(b). The dynamics of CA provide the necessary projection of the input data onto an expressive and discriminative space-time volume that can be used as the feature vector. It was previously shown that the cellular automata reservoir holds a distributed representation of high order attribute statistics [18]. Thus, 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.

Figure 2 shows more details for ReCA algorithm, where the initial state at time n, $A_0^{(n)}$ is evolved using a pre-specified ECA rule from 1 to I iterations to obtain the CA evolution states $(A_1^{(n)}, A_2^{(n)}, \ \ldots, A_I^{(n)})$ (n varies from 1 to T, the input sequence length). XOR[4] operation is used to insert the new input sequence X_n in the reservoir as follows: $A_0^{(n)} = A_I^{(n-1)} \oplus X_n$ and shown in Figure 2. Then the CA states $(A_1^{(n)}, A_2^{(n)}, \ \ldots, A_I^{(n)})$ are concatenated to obtain a single state vector $A^{(n)}$ that will be used as a feature vector with length of $L_{CA} = IL$ to

[2] The value of R_i should be chosen carefully to reduce the interference between non zeros that have different locations in consecutive time steps.

[3] The rotation right is to reduce the interference between non zeros that have the same location in consecutive time steps.

[4] *XOR* computes the correlation, which provides a lossless merging of two binary numbers: it outputs 1 if something different from cell content is presented to the cell.

estimate the output at time step n ($y(n)$) using linear regression in read-out stage.

2.3 Read-out stage

In this stage the cellular automaton state vectors (feature vector with dimension of L_{CA}) are used in linear regression to estimate the output:

$$y(n) = W_{out} * A^{(n)} \tag{1}$$

In our experiments, there are two cases for the output: 1. There is only one output y at last time step $n = T$. 2. There is an output for each time step. The size of the feature space and matrix W_{out} differs in the two conditions. See [10] for details.

3 Covariance and Stack Representations

Two other representations with different levels of distributedness and computation are introduced for comparison with ReCA.

1.Covariance Representation: The operator C_k is defined as:

$$C_k = \Pi_k A_0 \oplus \Pi_{-k} A_0 \ , \tag{2}$$

where, Π_k and Π_{-k} are permutation matrices $+k$ and $-k$ shifts and $\oplus$ is bitwise XOR. C_k computes the pairwise covariance of the input attributes as in tensor products [13] and memorizes those for each sequence input. In this representation the operator C_k in (2) is computed in the reservoir to produce the covariance evolution states, instead of applying ECA rules. Second order statistics information is **locally** held in this representation.

2. Stack Representation: In this representation there is **no computation** involved, the input sequence steps are reserved one after another in raw format. T memory blocks of size L_{in} are used as the feature space. It can be considered as the simplest feedforward formulation (no interference between time steps), thus it is not a fixed length representation as in recurrent architectures, which is problematic for large T and L_{in}.

3. Hybrid Covariance and Stack Representations: Instead of using the raw input of the sequence in Covariance and Stack, we can first apply cellular automata nonlinearity and project the input feature space onto the cellular automata state space. This provides a hybrid feature space, in which a minimal amount of computation and distributedness of cellular automata state space is injected into the Covariance and Stack representations.

4 Experiments

In the experiments we trained linear regressors using three representations of sequence; ReCA, Covariance and Stack, on various pathological synthetic tasks.

A set of N_{train} (No of training examples) and N_{test}[5] (No of test examples) input time series and their associated outputs (Target) are synthesized for each task [6]. The experiments target is to achieve a **zero test error** (there should be **no** False Bits in the predicted output). The parameters I (No of CA evolution iterations), R_i (No of bits that represent each input attribute) and N_{train} (No of training set examples) are varied to find the minimal configuration to achieve zero error. Normal equation based linear regression implemented by pseudo-inverse is used in read-out stage.

The classical *Pathological Synthetic Tasks* are used in order to test short-term-memory capability of representations. These tasks have been proposed by Hochreiter and Schmidhuber in [4] that all exhibit pathological long term dependencies and known to be *effectively impossible* for gradient descent [11,7] based pure feedforward architectures. These tasks can be classified into 3 categories: **1.** Memory tasks (5 bit, 20 bit and Random permutation), **2.** Temporal order tasks (2 and 3 symbols) and **3.** Arithmetic or logic operation tasks (XOR, Addition and Multiplication) [6]. We refer the reader to [6] and [10] for a detailed description of them. The difficulty of these tasks increases with the number sequence time steps T, because longer T exhibits longer range temporal dependencies. All inputs in the tasks are originally one hot encoded (i.e. one nonzero entry per time step). In order to make the input more appropriate for real applications, *Binary Encoded* versions are used. As an example, we illustrate the change of 4 bit one hot encoded input to 2 bit binary encoded input: 0001 $\Rightarrow$ 00, 0010 $\Rightarrow$ 01, 0100 $\Rightarrow$ 10 and 1000 $\Rightarrow$ 11. [7]

5 Results and Discussion

The results of our experiments are illustrated in Table 1. ReCA framework has solved all pathological tasks with zero error either **a.** *directly*, as in random permutation, 5 Bit and temporal order tasks, or **b.** by *expanding* the input using R_i (reduces the interference between input bits) for 20 bit task, or using **c.** *multi-layer* CA expansion for addition, multiplication and XOR tasks. Covariance and Stack representations are capable of solving only memory tasks. However, Stack also fails in binary encoded 20 bit memory task, which implies incapability for more realistic applications. Therefore, we conclude that ReCA representation is superior due to its distributed representation (more robust to interference [8] due to activity merging) and computation of higher order attribute statistics.

---

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

[6] In addition and multiplication tasks the input is binary, in future work decimal numbers will be used after binarization.

[7] 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).

[8] Interference can be defined as the modulation of reservoir activity with injection of input at each time step that disturbs one-to-one correspondence between the input sequence and the reservoir activity due to the sequence.

Table 1. Results for Pathological tasks using the proposed representations. The last column (red and bold) is the number of false bits in predicted output of test set. The other columns are for various parameters. T is the sequence length of task input, I is the iterations of CA evolution, N_{train} is the number of training example, $\%N_{train}$ is the ratio between training examples and all input possibilities, f is the expansion ratio, R_i is the number of bits to represent each bit of task input, No of False Bits is the number of false bits in predicted output.

Pathological Task		Method	Multilayer ECA rule Expansion*	T	I	Ntrain	% Ntrain	f	Ri	No of False Bits
Memory	5 Bit	ECA Rule 90	-	1000	4	32	1	1	1	**0**
	20 Bit Normal	ECA Rule 90	-	300	20	120	1.23E-05	1	2	**0**
		Covariance	-	300	19	120	1.23E-05	1	3	**0**
		Stack	-	1000	-	5	5.12E-07	-	-	**0**
	20 Bit Binary Encoded	ECA Rule 90	-	200	24	250	2.56E-05	1	8	**0**
		Covariance	-	100	19	250	2.56E-05	1	2	**0**
		Stack	-	100	-	500	5.12E-05	-	-	**47**
	Random Permutation	ECA Rule 90	-	1000	2	200	$\cong 0$	1	1	**0**
		Covariance	-	1000	2	200	$\cong 0$	1	1	**0**
		Stack	-	1000	-	1200	$\cong 0$	-	-	**0**
Temporal Order	2 Symbols	ECA Rule 150	-	500	8	6000	1.20E-02	1	1	**0**
		Covariance	-	200	8	6000	7.54E-02	1	1	**151**
		Stack	-	200	-	6000	7.54E-02	-	-	**4**
		ECA Rule 150	122 (1)	1000	8	8000	4.00E-03	0.5	1	**0**
		Covariance	122 (1)	50	8	1500	3.06E-01	1	1	**56**
		Stack	122 (1)	500	-	14000	2.81E-02	-	-	**0**
	3 Symbols	ECA Rule 90	-	50	24	7000	4.46E-02	1	1	**0**
		ECA rule 90**	-	200	16	2000	1.90E-04	1	1	**0**
Arithmetic and Logic Operaton	Addition	ECA rule 150	40 (1)	500	20	2500	$\cong 0$	0.5	1	**0**
		Covariance	110 (1)	50	20	3300	2.39E-15	1	1	**6**
		Stack	110 (2)	1000	-	3500	$\cong 0$	-	-	**0**
	Multiplication	ECA rule 150	110 (2)	500	12	2000	$\cong 0$	0.5	1	**0**
		Covariance	110 (2)	50	12	2000	1.45E-15	1	1	**17**
		Stack	110 (2)	1000	-	14000	$\cong 0$	-	-	**0**
	XOR	ECA rule 150	40 (1)	1000	4	500	$\cong 0$	0.5	1	**0**
		Covariance	110 (2)	50	4	3200	2.32E-15	1	1	**35**
		Stack	110 (2)	50	-	3200	2.32E-15	-	-	**39**

* The number between brackets is the number of iterations are used in multilayer CA.
** The feature vector for regression consists of ***last*** CA evolution state from ***all*** time steps .

For hybrid representations of Stack and Covariance, we observe that Stack becomes capable of solving addition, multiplication after cellular automata input expansion, but **not XOR task**. This is possibly due to requirements of the nonlinear nature of XOR task. However, Covariance representation benefits very little from the initial CA expansion. Multilayer CA expansion with non linear rules is a very powerful technique in general, because it also enables ReCA to solve XOR, Addition and Multiplication tasks and improve temporal order tasks. In ReCA, linear (*additive*) ECA rules (90 and 150) are essential for the reservoir evolution, to achieve lossless injection of input at each time step [9]. However, non linear ECA rules (for rule 40 see [2]) should exclusively be used in multilayer

[9] Linearity maximizes one-to-one correspondence between input sequence and the reservoir activity due to the sequence.

expansion as shown in the fourth column of Table 1. Please see more results in [10].

Comparing ReCA with previous approaches, it outperforms: **1.** Martens and Sutskever (2011) [11] and Pascanu et al. (2013) [12] in sequence length T, in their studies the zero test error has been obtained for T ranging from 50 to 200, but in our experiments T ranging from 200 to 1000. **2.** Jaeger (2012) [6], where the zero test error could not be achieved in 20 Bit binary encoded task using ESNs. ReCA outperforms ESN in computational complexity for most of the tasks as listed in Table 2. There is more than **100**× speedup/energy savings for memory tasks and **8**× for temporal order tasks, but ESN is **2**× faster for XOR task. Comparing the representations of ReCA and ESN, the computation performed in the ReCA is much more transparent for analysis and improvement compared to ESNs, in which the state evolution is untraceable due to random and irregular distributivity.

Table 2. Number of bitwise operations for ESN and ReCA frameworks for solving some pathological tasks.

Task	*ESN* (Floating Point⇒Bit)	*CA* (Bit)	Speedup
20 Bit, $T_d = 200$	105.6 M⇒3380 M Bit	24.8 M Bit	136X
3 Symbols, $T = 200$	5 M⇒160 M Bit	20.5 M Bit	7.8X
XOR, $T = 1000$	0.2 M⇒6.4 M Bit	16 M Bit	0.4X

6 Conclusion

Cellular Automata Reservoir framework (ReCA) constructs a novel bridge between computational theory of automata and recurrent neural architectures. We show that the ReCA achieves zero error in all pathological synthetic tasks of sequence learning. Sequence learning is an essential capability for a wide collection of intelligence tasks such as language, continuous vision (i.e. video), symbolic manipulation in a knowledge base etc. ReCA outperforms the Covariance and Stack representations because it enables both computation of high order statistics and distributedness, where Stack representation provides pure memorization, and Covariance representation computes only second order statistics.
Usage of cellular automaton instead of real valued neurons in reservoir computing framework greatly simplifies the architecture, makes the computation more transparent for analysis, and provide enough computation for sequence learning even though it is a very sparse network with identical binary units. Increasing cellular automata iterations I, and the size of each input entry R_i, reduces interference in the reservoir in a predictable manner, making the computation more similar to a feedforward architecture (i.e. Stack representation).

Acknowledgments. This research is supported by The Scientific and Technological Research Council of Turkey (TUBİTAK) Career Grant, No: 114E554.

References

1. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on 5(2), 157–166 (1994)
2. Braga, G.: Chaotic properties of the elementary cellular automaton rule 40 in wolframs class i. Complex Systems 17, 295–308 (2007)
3. Doya, K.: Bifurcations in the learning of recurrent neural networks 3. learning (RTRL) 3, 17 (1992)
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
5. Jaeger, H.: The echo state approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report 148, 34 (2001)
6. Jaeger, H.: Long short-term memory in echo state networks: Details of a simulation study. Tech. rep., Technical Report (2012)
7. Kremer, S.C., Kolen, J.F.: Field Guide to Dynamical Recurrent Networks. Wiley-IEEE Press (2001)
8. LukošEvičIus, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Computer Science Review 3(3), 127–149 (2009)
9. Maass, W., Natschläger, 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)
10. Margem, M., Yilmaz, O.: An experimental study on cellular automata reservoir in pathological sequence learning tasks (2016), `http://ozguryilmazresearch.net/Publications/MargemYilmaz_TechReport2016.pdf`, [Online; accessed 31-March-2016]
11. Martens, J., Sutskever, I.: Learning recurrent neural networks with hessian-free optimization. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 1033–1040 (2011)
12. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13) (2013)
13. Smolensky, P.: Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artificial intelligence 46(1), 159–216 (1990)
14. Verstraeten, D., Schrauwen, B., dHaene, M., Stroobandt, D.: An experimental unification of reservoir computing methods. Neural Networks 20(3), 391–403 (2007)
15. Wolfram, S.: A New Kind of Science, vol. 5. Wolfram media Champaign (2002)
16. Wolfram, S., et al.: Theory and applications of cellular automata, vol. 1. World scientific Singapore (1986)
17. Yilmaz, O.: Analogy making and logical inference on images using cellular automata based hyperdimensional computing. In: NIPS, Workshop on Cognitive Computation (2015)
18. Yilmaz, O.: Machine learning using cellular automata based feature expansion and reservoir computing. Journal of Cellular Automata 10(5-6), 435–472 (2015)
19. Yilmaz, O.: Symbolic computation using cellular automata based hyperdimensional computing. Neural Computation 27(12), 2661–2692 (2015)