Classification and Computation on Non-uniform Finite Cellular Automata Networks

Hürevren KILIÇ[1] and Marifi GÜLER[2]

[1] Bilkent Computer Center, Bilkent University, Bilkent, 06533 Ankara, Türkiye

[2] Department of Computer Engineering, Middle East Technical University, 06531 Ankara, Türkiye

Abstract

In this paper, a model called *Non-uniform Finite Cellular Automata (CA) Network* is introduced, and its classification and computation power studied. The suggested model is similar to the original Cellular Automata Network model, with its local neighborhood property, but neighborhood definitions of cells are not the same (non-uniform) for each cell and determined by an algorithm. The model is similar to the Neural Network (NN) model with its different local cell (neuron) transition function definitions and with its training (or feature extraction) mode. Depending on the nature of input-output templates, computation can be done on the system. Necessary and sufficient conditions for doing computation on this model are based on the work of Tchuente, and will be elaborated upon in this paper.

1. INTRODUCTION

An Automata Network may be defined as a locally connected large set of cells (finite automata), which can evolve at discrete time steps through mutual interactions. In this model, space, time and cells are discrete. Automata networks have many application areas such as, Artificial Intelligence, Pattern Recognition and Learning Systems.

A particular class of Automata Networks is the Cellular Automata (CA) Networks. In computer science cellular automata is used to model parallel processing and Von Neumann (*self-reproducing*) machines. In cellular automata, space is divided into discrete small units called *cells* or *sites*. Each cell can take k different state values. At time t, all the cells will have a specific state value. Rules local to a specific cell determine what the value of that cell at time $(t+1)$ will be. Rules are the same for each cell. For k number of states per cell and n number of cells in the neighborhood of a cell, k^z (where $z = k^n$) possible local transition functions for that cell exist.

Another important class of Automata Networks, called Neural Networks (NN), with similar characteristics to CA Networks, use only threshold type functions as their neural transition functions. Unlike cellular automata, in neural networks each neuron does not have to evolve according to the same neural transition function. However, in general, neural networks without the neighborhood concept that exists in cellular automata model, require great number of neural connections (or neural dependency) between their neurons (e.g. Hopfield model, multi-layer perceptron model).

2. AUTOMATA NETWORKS

An automata network can be defined as locally interconnected set of cells, which can evolve at discrete time steps. This evolution occurs through mutual interactions between these locally connected cells. Formally, an automata network can be described as mapping F from S^n into itself, where S is finite state space and n is the number of inter-connected cells. The structure of connection is determined by F; if i^{th} component of mapping F depends on j^{th} variable, cell i receives a connection from cell j. A state of the network is a vector X in S^n and dynamics on the network can be defined as:

$Y = F(X)$ where F : mapping function $X, Y \in S^n$ are state vectors

At each time step, each automaton in cell i computes its next state according to the rules of mapping F_i, and causes a global evolution of network. This is known as parallel iteration. Other iteration modes such as sequential mode where cells are updated in a prescribed order and memory mode where previous cell values are used, also exist [1].

Since the state space S is usually finite, at the end of at most k^n steps of evolution, (where k is the number different states that a cell can be in and n is the number of cells), the system will enter into a cycle or a fixed point which can be accepted as a cycle of period 1. The mapping function F defined above is deterministic, i.e., we can guess exactly the next state of the system from its present state. In this paper, only the *deterministic* (not random) automata networks are considered and studied. Automata networks are discrete and dynamical systems in time and space and they can be represented by a graph, where each node of the graph takes one of the states in a finite set. Moreover, the evolution of the network results from changing states of each site (or node) according to a transition rule that takes into account only the state of its neighbors in the graph.

An important class of Automata Networks are *Neural Networks*. In this model, the graph representing the network is non-oriented, i.e., direction between nodes (neurons) is not important and graph is finite. Nodes can take one of two state values { -1,1 }. The transition rule is a threshold function whose inputs are the output of other threshold units (neurons) weighted by real numbers. In general, the sign threshold function is used to calculate the new state of a neuron. If the weighted sum of neuron values, other than the currently calculating neuron, is positive or zero, then the next state of that neuron is 1, otherwise it is -1. Moreover, in Neural Networks there is no restriction on site updating mode.

Cellular Automata Networks also constitute a particular class of Automata Networks and were originally introduced by Von Neumann. Capabilities and limits of this model is one of the topics of interest of this paper. In this model, the neighborhood and the transition rules are the same for all sites (or cells). Site updating mode is synchronous.

3. THE SUGGESTED MODIFIED CELLULAR AUTOMATA MODEL

"Is it possible to construct a model similar to cellular automata and neural network models, that has the capability of classification and computation, but with fewer number of dependency (connections) between its cells (or neurons) ?".

The main difference between the suggested modified CA model and the original CA model concerns neighborhood definition (See Figure 1). In the original CA model, each cell has the same neighborhood definition; however, in the suggested model each cell does not have to have the same neighborhood definition.

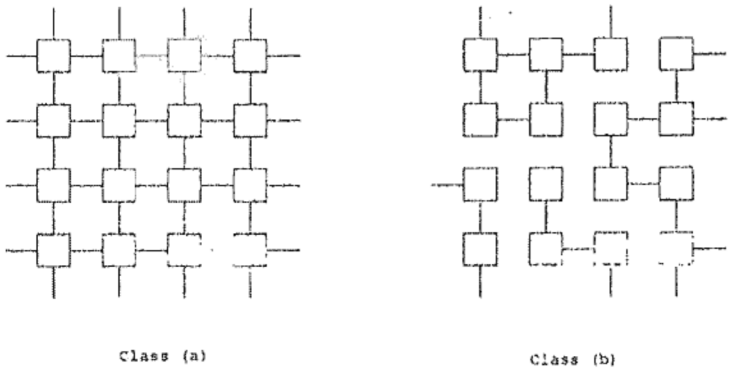

Figure 1. The original CA model (a) and the suggested CA model (b) in two-dimensional cellular space.

Neighborhood degree of a cell can be defined as the number of neighbors that this cell accepts state values to specify its next state. In the original CA model each cell has the same transition function definition, but in the suggested CA model only the cells having the same neighborhood degree have to evolve according to the same transition function rules. In other words, different neighborhood degrees may cause different transition functions. In the suggested CA model, there is no distinguished state called *quiescent state* which causes a cell to stay at quiescent state if all of its neighbors are in quiescent state in the original model. The transition functions of the modified model do not have to have such a distinguished state. In the suggested model, there is no initially set neighborhood structure of cells. Neighborhood of each cell is determined according to the nature of input and output patterns desired to be mapped. In the original model, the neighborhood structure of cells is static and fixed.

The modified CA model has two modes of operation similar to NN model: training and classifying (or mapping). In the training mode, neighborhood degree and transition functions of each cell are determined using input-output template pairs. Once cell functions are defined, one can introduce any input template to the system and get its intended output template. In a sense, the whole cell system can be considered as a classifier. Different from NN systems, the suggested CA system has no error correction capability or fault tolerance property. In NN systems, similar input patterns may converge to the same output patterns and this property is very important and useful in recognition and classification systems. However, in the proposed model, since the transition functions are defined as mapping but not as inequalities (threshold function in Neural Nets), similar input patterns cannot be guaranteed to converge to the same output pattern. Patterns other than the input patterns used during training may be not mapped. For such unmapped patterns, a state 'X' is introduced in order to complete the definition of local transition functions. If such a pattern that is not mapped during training mode is extracted from the input template the next state of current cell for that pattern becomes 'X'. Therefore, a cell in the suggested model can be in one of three states {0, 1, X}.

One of the famous neural network models is the Hopfield's Net model. In this model, neurons are assumed to be fully-connected, i.e., each of n neuron is connected to and gets input from the other $(n-1)$ neurons. Thus, each neuron is functionally dependent on every other neuron. Especially for hardware implementations of neural networks, decreasing the number of neural connections in nets is an important problem. In the suggested cell system, each cell depends on only its neighbor cells and the state of other cells need not be considered. If the input patterns are very similar to each other, neighborhood degree of cells increases and as a result, the system's performance decreases. On the other hand, input patterns having different characteristics i.e. different bit sequences decrease the degree of neighborhood and increase the system's performance.

The modified CA system makes its decision in one step of evolution similar to Perceptron and Kohonen Neural Network models. In some other NN models, such as Hopfield Model, system evolves until it converges to a stable configuration and the resulting configuration is the system's decision on the introduced input pattern.

The suggested cell model can simply be considered as a parallel mapping system. It maps a given input template to an output template, both introduced during training mode, in parallel. Pseudo code of algorithm used to construct the modified CA model for introduced input-output templates is as follows:

1. Get input and output templates
2. Set number of cells in the cell system equal to the number of bits in one input (or output) template
3. Set neighborhood degree of each cell to 1
4. While there are more cells to be processed
    4.1. While there are more templates to be processed
        4.1.1. Extract bit pattern of current cell from the current input

template being processed which is constituted by itself and its current neighbors

4.1.2. If this pattern with degree of neighborhood i is mapped to a different output bit before

4.1.2.1 Increase neighborhood degree i of current cell by 1

4.1.2.2 Go to step 4.1.1

4.1.3. If this pattern is not mapped to any output bit before

4.1.3.1 Add input pattern and its output bit into look-up table as a rule for cell transition function of current neighborhood

4.1.3.2 Skip to the next template

4.1.3.3 Go to step 4.1

4.1.4. If this pattern is mapped to the same output bit before

4.1.4.1 Skip to the next template

4.1.4.2 Go to step 4.1

4.2. Skip to next cell

4.3. Go to step 4

5. Now, the neighborhood degree and the transition function of each cell is determined. Read input template desired to be mapped

6. For each cell of input template apply its cell function determined at previous steps, and find output bit of that cell. Next state of cells constitute the output template produced by the system

The method to extract an input pattern from an input template for a cell with current neighborhood degree k, is to first take the bit at current cell position and continue taking bits one from the right and one from the left until the current neighborhood degree k is reached (See Figure 2).

| ..... | .... | 7 | 5 | 3 | 1 | 2 | 4 | 6 | 8 | ..... | ..... |

Figure 2. The order of bit extraction from an input template, for k = 8

Input templates are assumed to be one-dimensional and circularly connected, i.e. the left neighbor of the first cell is the last cell, and the right neighbor of the last cell is the first cell. Also, the length of an input template is equal to the length of the output template.

An important question should be asked here: "Does the algorithm guarantee that the program will not enter into an infinite loop and will not increase the neighborhood degree to infinite?" Confliction case between two input patterns may occur if their output bits are different. Two different types of confliction may occur. One is *external conflict* and the other is *internal conflict*. External conflict occurs when the output bit of the current cell conflicts with the output bit of the input pattern of a different template. External conflict does not cause an infinite loop. Since the input templates that are introduced are guaranteed to be different from each other (that is, one input template can only be mapped to a unique output template), one can guarantee that the conflict case will certainly be resolved when neighborhood degree of the cell is the length of the input pattern (i.e. input pattern = input template).

Internal conflict occurs when input patterns of conflicting output bits on table are patterns of the same input template. Since the input templates of conflicting output bits are the same, confliction cannot be resolved as it can be in external conflict when neighborhood degree of cell reaches its maximum value. Instead, this problem can be solved by concatenating the output bit of the cell to the end of the input template, making two input templates different, and adding it into table with its output bit. Therefore, the maximum number of neighbors that a cell can have is $(n+1)$, where n is the length of

the input template, and it is reached when successive internal conflicts occur. Here, $+1$ in $n+1$ comes from the output bit of the cell. As a result, we can say the algorithm guarantees that the program will not to enter into an infinite loop, because both external and internal conflictions are resolved.

4. COMPUTATION ON THE SUGGESTED CELLULAR AUTOMATA MODEL

"Computation can be done in infinite uniform cellular automata structures" [6]. In such structures, the number of cells in automata is infinite and the neighborhood structure should be local and regular i.e. the same for all cells. However, in finite structures of cellular automata networks these restrictions are no longer considered and such a network can be defined as triple:

N = (G, Q, F) where

G : a directed graph of order n, representing the interconnection of vertices (or cells)

Q : the finite non-empty set which represents the set of states that cells can assume

F : a collection of functions from Q^n (n is the number of cells in structure) into itself, representing the set of possible *global transition functions* of the networks. The *global transition function* is comprised of the local transition functions of individual cells and determines the network's global behavior.

Each cell of the automaton is represented by a vertex on a graph. If a cell i has neighbor j (i.e. i is dependent on the state of j at time t in order to determine its state at time $t+1$), then there is an arc from vertex j to i on the graph. An example graph of the suggested modified automata model can be given. Assume that we have 6 bit length input-output pairs, and at the end of the training mode, neighborhood degree of cells from 1 to 6 are determined as 3,2,2,4,5,6. The resulting graph will be as in Figure 3.

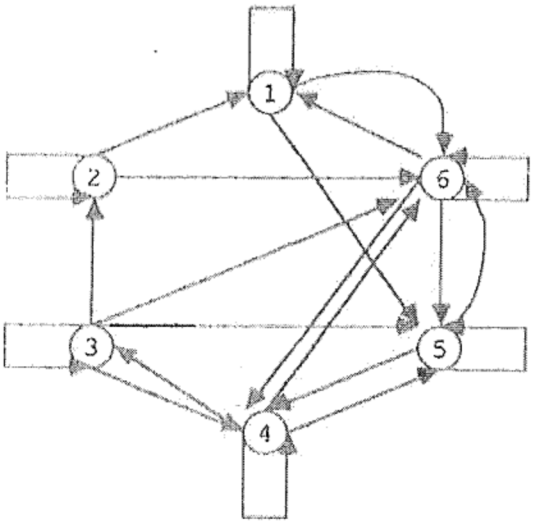

Figure 3. Graph of an example modified cellular automaton model

Now, it may be asked "Is it possible to do computation on the suggested model?" In networks of the form N = (G, Q, $A(Q^n)$) where N is a network with arbitrary graph connections, G and Q are as explained above, and $A(Q^n)$ is the set of mapping from Q^n into itself, the computable functions are characterized by the following theorem proposed by Tchuente:

Theorem: For any finite set Q of cardinality greater than one, $A(Q^n)$ is computable on a network $N = (G, Q, A(Q^n))$ of order n, if and only if G is strongly connected and contains a vertex v_r such that, for any vertex $v_i <> v_r$ (v_i, v_r), is an arc of G.

Proof of the theorem can be found in [6]. This theorem can help us determine the conditions under which one can do computation on the suggested modified cellular automaton. In the suggested model $Q = \{0, 1, X\}$ and its cardinality is 3 which is greater than 1. $A(Q^n)$ depends on the nature of input-output templates and is a mapping from Q^n to itself. *n* is the number of bits of one input or output template. However, one cannot guarantee the existence of a cell having neighborhood degree n, in other words, a vertex having arcs from each of the other vertices into itself or *strong component*, for given input-output pairs. Therefore, the answer to the question "Is it possible to do computation in the suggested non-uniform finite cellular automata network?" is, *not always*. This is because the topology of the graph of automaton network depends on the nature of input-output templates.

Example:
Assume that we have the following input-output templates:

Table 1
Sample input and output templates.

Pair #	Input Template	Output Template
1	01000	11110
2	00000	11100
3	01101	11010
4	10001	10100
5	10110	10101

The neighborhood degree of cells from 1 to 5 at the end of the training mode are 1, 3, 2, 5 and 4, respectively. As can be seen from the graph of the example automata network (Figure 4), computation can be done on this cellular structure, because the fourth vertex has arcs from all other vertices.

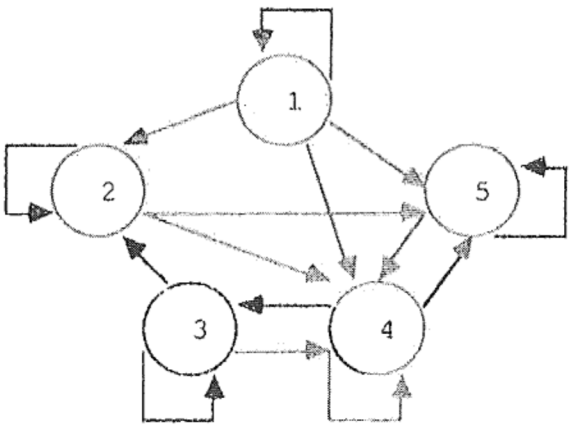

Figure 4. Graph of a one-dimensional modified cellular automaton.

5. CONCLUSION

In this paper, a modified cellular automata model capable of doing classification (or mapping) is introduced. Also, it is shown that one can do computation on this model depending on the nature of input and output templates. At the end of this study, we constructed a model that can map a given input template to its corresponding output template, that were both introduced during training (or feature extraction) mode of the system. The model suggests a different way of representation of information. It converts input and output templates into a form constituted by a look-up table and a neighborhood array. Look-up table keeps n-to-1 mappings showing the next state of cells. The neighborhood array contains the neighborhood degree of each cell in the automaton. This kind of representation of information does not provide an efficient method for storage of information. However, it is in a form that can suitably be used for parallel processing. The larger the size of look-up table, the longer the time to search it. By means of classical searching methods, look-up table search time can be decreased up to a degree.

An alternative to look-up table can be functional representation of it. One can find a function for each neighborhood degree. This makes the model more storage efficient and eliminates the need for table look-up that decreases the time efficiency of the model. A method for finding these functions could be the use of neural networks. For example, since the look-up table contains n-to-1 mappings for each neighborhood degree, multi-layer perceptron model of neural nets can be suitably applied to each of the different neighborhood degrees. The use of neural nets in combination with the suggested model increases the training time, especially if multi-layer perceptron is trained by the gradient back-propagation method, but decreases the system's classification (or mapping) time.

An interesting property of the suggested model is that computation can be done on it if it has a cell whose neighborhood degree is equal to the length (in bits) of a template, depending on input and output templates. In other words, if there exists a cell which is functionally dependent on all of the other cells in the automaton, computation can be done on this automaton.

6. REFERENCES

[1] Goles, E., Martinez, S., *Neural and Automata Networks* Kluwer Academic Publishers, Netherlands 1990.

[2] Hopfield, J.J., *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, Proc. Nat. Acad. Sci. USA, 79(1982) 2254-2258.

[3] Kılıç, H., *Non-uniform Cellular Automata Networks Having Capability of Classification and Computation,* Master's Thesis, Middle East Technical University, 1992.

[4] Lippmann, R.P., *An Introduction to Computing with Neural Nets*, IEEE ASSP Magazine April 1987.

[5] Packard, N.H., Wolfran, S., *Two-dimensional Cellular Automata*, Journal of Statistical Physics, Vol 38 901-946 1985.

[6] Tchuente, M., *Automata Networks in Computer Science*, Princeton Univ. Press Princeton, New Jersey 1987.

[7] Von Neumann, J., *Papers of John von Neumann on Computing and Computer Theory, edited by W. Aspray and A.W. Burks*, Cambridge, Mass. : MIT Press 1987

[8] Wolfram, S., *Universality and Complexity in Cellular Automata*, Physica 10D 1984 1-35.