INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

Continuous Function Identification with Fuzzy Cellular Automata.

Bohdana Ratitch

School of Computer Science

McGill University, Montreal

May, 1998

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements of

the degree of Master of Science.

Abstract

Thus far, cellular automata have been used primarily to model the systems consisting of a number of elements which interact with one another only locally; in addition these interactions can be naturally modeled with a discrete type of computation. In this thesis, we will investigate the possibility of a cellular automata application to a problem involving continuous computations, in particular, a problem of continuous function identification from a set of examples. A fuzzy model of cellular automata is introduced and used throughout the study. Two main issues in the context of this application are addressed: representation of real values on a cellular automata structure and a technique which provides cellular automata with a capacity for learning. An algorithm for encoding real values into cellular automata state configurations and a gradient descent type learning algorithm for fuzzy cellular automata are proposed in this thesis. A fuzzy cellular automaton learning system was implemented in a software and its performance studied by means of the experiments. The effects of several system's parameters on its performance were examined. Fuzzy cellular automata demonstrated the capabilities of carrying out complex continuous computations and performing supervised gradient based learning.

Résumé.

Les automates cellulaires ont jusqu'ici été utilisés essentiellement pour modéliser des systèmes consistants en éléments qui communiquent les uns avec les autres de façon locale. De plus, ces communications peuvent être modélisées naturellement à l'aide de calculs discrets. Dans cette thèse nous nous intéressons à la possibilité d'appliquer les automates cellulaires à un problème inpliquant des calculs continus, et particulièrement au problème d'identification d'une fonction continue à partir d'ensemble d'exemples. Un modèle d'automate cellulaire "flou" (fuzzy) est introduit et utilisé tout au long de notre recherche. Deux questions sont d'un intérêt particulier dans le contexte de ce problème: la représentation des valeurs continues dans la structure d'un automate cellulaire, et l'élaboration d'une technique d'apprentissage applicable aux automates cellulaires. Un algorithme pour encoder des valeurs continues dans les automates cellulaires est proposé, ainsi qu'un algorithme d'apprentissage de type "gradient descent". Un prototype basé sur un automate cellulaire flou a été implementé en logiciel, et sa performance a été étudiée au moyen d'essais. Les effets de plusieurs des paramètres du système sur sa performance ont été étudiés. Les automates cellulaires flous ont démontré leur capacité à exécuter des tâches complexes de calcul continu, et à accomplir des tâches d'apprentissage supervisé.

Acknowledgments.

In the first instance it is my pleasure to express my deepest gratitude to my thesis supervisor Denis Térien. His guidance and opinions have assisted me through my entire research work and have in great measure made this thesis possible. His constant encouragement and support have helped me tremendously in my integration into the Canadian academic and research community. In addition I extend my sincerest appreciation and gratitude to Gordon Broderick from Noranda Technology Center. This thesis has greatly benefited not only from his expertise but also from our frequent discussions which were a constant source of inspiration and encouragement.

To Ricard Gavalda I am very grateful for his many valuable ideas and comments. I would also like to acknowledge Hector Budman for the interesting and helpful discussions.

To Jean Sebastien Bolduc I extend my expression of thankfulness for his cooperation and support in the many situations of our lives as fellow students.

I am grateful to my husband, Andrij Serbyn, for his editorial assistance and especially for his patience, encouragement and belief in me.

Many thanks to all the professors and instructors whose knowledge I was able to share while taking courses at McGill's School of Computer Science and particularly to all the members of our school staff for their work which provided an excellent environment and atmosphere conducive for study and research.

I am grateful to Noranda Inc. for the financial support provided for this research.

Contents

Chapter 1

Introduction.

Cellular automata represent one of the computational models which has been attracting the attention of researchers in theoretical computer science as well as of scientists in other disciplines, who applied it in numerous ways. Cellular automata have proven to possess the capabilities of a universal machine which is an important feature, at least from a theoretical perspective. Being applied to various practical problems, cellular automata also exhibit very useful and interesting characteristics. The most notable of them are massive parallelism, locality of cellular interactions and simplicity of basic components. Cellular automata perform computations in a distributed fashion on a spatial grid. In this they differ from a standard approach to parallel computations whereby a problem is split into independent sub-problems later to be combined in order to yield a final solution. Cellular automata suggest a new approach in which a complex global behavior can be modeled by non-linear spatially extended local interactions.

Thus far, cellular automata have been used primarily to model the systems consisting of a number of elements obeying identical laws of local interactions (e.g. problems of fluid dynamics, crystal growth, tissue engineering, socio-environmental problems, etc.). In other words, information about these systems could be easily mapped onto cellular automata's discrete computational space. The possibilities to

apply cellular automata to other systems and problems, that do not possess this characteristic, have not yet been well explored. It is an accepted point of view that cellular automata provide a natural and efficient tool for discrete types of computations. But computational capabilities of cellular automata in a broader sense, including continuous computations, still require additional study.

The wide applicability of cellular automata is limited because the methodologies for designing cellular automata, intended to solve specific pre-defined tasks, are still underdeveloped. Such designing techniques would be extremely useful since there exist many problems for which local interactions, that would drive cellular automata to solve these problems, are not known in advance. Some work has already been done in this area, mainly using genetic algorithms to evolve cellular automata transition rules. Some algorithms were proposed to derive the transition rules on the basis of available history of the cellular automata global states. To the best of our knowledge, only one different approach has been considered: reinforcement learning for probabilistic cellular automata [41], [11]. Successful results of these studies suggest that other learning concepts, that were developed in different areas of machine learning, can also be efficiently applied to cellular automata.

The present study pursued two main objectives:

- *To validate the capacity of cellular automata to perform complex continuous computations.* The successful results in this aspect broaden the area of potential applications for cellular automata.
- *To assess cellular automata abilities and efficiency in gaining knowledge in a process of learning.* Development of new learning techniques, designed specifically for cellular automata, facilitates their application in those areas where such an application is theoretically possible, but an exact description of a suitable cellular automaton is not known.

To achieve the above stated objectives, cellular automata were applied to a problem of function identification. The problem of function identification is undergoing

constant development in several scientific domains. This problem involves approximation or modeling of an unknown function using a set of available input-output pairs of this function. The function identification problem has become especially acute in today's information loaded society. Technology has developed various means of data acquisition and storage, whereas the techniques for processing available information and converting it into useful knowledge are still far behind. An ability to model and predict different natural and man-made systems and processes is highly desirable in many spheres of science and industry. New approaches to function identification has recently arisen in the field of artificial intelligence. Artificial neural networks are the most popular amongst them and have already proved to be quite successful. But a search for new computational models and algorithms capable of performing the identification task is still very active. Cellular automata have never been applied yet in the context of this problem, but their powerful computational features suggest the possibility of their good performance in this area.

The idea of cellular automata application to the problem of function identification involves two major issues: knowledge representation and acquisition. If real valued functions are to be approximated by cellular automata models, real values have to be represented in some way on a cellular discrete structure. Once this issue is resolved, specific techniques to generalize knowledge, present in a set of known input-output pairs, have to be designed specifically for a cellular automata type of computation. In other words, the methods that would enable a cellular automaton to learn an unknown function from a set of examples have to be discovered or existing learning algorithms must be adapted to the cellular automata model.

Most function identification approaches rely on parametric models, i.e. computational models used to approximate functions are assumed to have some (pre-defined) general form within which a model that approximates a specific function is defined by a specific set of parameters' values. One of the popular techniques to determine these specific values makes use of the gradient based optimization methods. In the present study, we explored the question of how these ideas, successful in applications

to other computational models, could be incorporated into the context of cellular automata.

We draw the reader's attention to chapters 2 through 5 where the background material and work associated with the matters explored in this thesis are presented. Chapter 2 is devoted to a cellular automata model of computation, its origin, characteristics and applications. The basics of fuzzy set theory are presented in Chapter 3. Learning paradigms used in today's machine learning theory are discussed in Chapter 4. Special attention will be paid to connectionist learning systems since the concepts employed in this field seem to be very suitable for the applications in the context of cellular automata learning. A review of the existing studies on cellular automata identification (learning) can also be found in Chapter 4. A problem of function identification is addressed in Chapter 5. In this chapter, a problem statement and review of the selected methods are presented. The remaining chapters of this thesis are devoted to our own research work and the results obtained in consequence of it. We will briefly acquaint a reader with the main issues addressed in these chapters.

The original model of cellular automata assumes that their basic components (or cells) are involved only into local logical computations, which are simple and fast. This is a valuable feature of cellular automata and we wanted to preserve it as much as possible. In order to enable cellular automata to acquire knowledge from examples it was necessary to pick out some of the defining elements of the cellular automata model, an exact description of which should be determined in the process of learning. The best choice seemed to fall on a cellular automaton state transition rule, which carries out most of the responsibility for the computations of an automaton. A gradient descent based technique, that was meant to be employed on the learning stage, would require a cellular automaton's transition rule to be differentiable. Taking all this into consideration, we found that the requirement of differentiability can be compromised while preserving local logical computations in the framework of fuzzy set theory. Following this idea, a definition of a fuzzy cellular

automata model will be proposed (see Chapter 6), in which a state transition rule assumes a form of parametric, differentiable fuzzy logic function. A gradient descent type learning algorithm was designed for this model to enable function identification based on a fuzzy cellular automaton learning system (see Chapter 7).

The issue of encoding real values into the cellular automaton state configurations is addressed in this thesis. An encoding problem is extremely important in this case, as it can directly influence both the precision of data representation and the complexity of a learning task. An encoding method was designed and is presented in Chapter 8.

It often happens in the field of machine learning that learning algorithms are difficult to analyze theoretically and proofs of their convergence or accuracy are difficult (if possible at all) to obtain. In this case, the system's performance can be assessed through empirical methods. Behavior of a fuzzy cellular automaton learning system and a learning algorithm designed for it was studied in detail through a series of experiments. The system's performance appeared to be influenced by the parameters that define a structure of cellular automata as well as by a learning rate parameter which is an attribute of a learning algorithm itself. Effects of these parameters are described in Chapter 9. The new learning system was tested on a variety of continuous functions and the observations concerning the system's performance are presented in this chapter as well.

The results obtained in this study are summarized in Chapter 10. They are positive and encourage further research in this direction. They demonstrate that cellular automata are capable of attaining good performance when applied to the problems involving continuous computations, specifically the problem of continuous function identification. Cellular automata once more demonstrated to be extremely interesting both from theoretical and applied points of view. The results of this study reinforce the idea that new applications for cellular automata should be sought and a variety of practical problems can benefit from their powerful computational features.

Chapter 2

Cellular Automata.

2.1 The Origin of Cellular Automata.

For the first time the idea of an automaton organized on a regular structure appeared in the studies of automata by von Neumann. Some of his work was published during his life [49],[47], while some was left in manuscripts, later to be edited and completed by Artur W. Burks in [13]. The development of von Neumann's theory of automata was motivated by his desire to improve the understanding of the natural systems (natural automata) as well as analog and digital computers (artificial automata). The main questions that he was concerned with were the following [13]:

1. *Logical universality:*

   - When is a class of automata logically universal?
   - Is any single automaton logically universal?

   This question had been answered by Turing, who showed that there is a universal Turing machine, which can perform any given computation. Then questions analogous to logical universality were posed about construction:

2. *Constructibility:*

   - Can an automaton be constructed by another automaton?
   - What class of automata can be constructed by a single specific automaton?

3. *Construction universality:*

   - Is any single automaton construction universal?

4. *Self-Reproduction:*

   - Is there a self-reproducing automaton?
   - Is there an automaton which can both reproduce itself and perform other tasks?

As a basis for his constructing automaton, Von Neumann used a simple automaton that possesses two stable states: *excited* and *quiescent,* corresponding to basic logical values of *true* and *false* . This was an extremely simplified model of a biological neuron and was called an *idealized* neuron. He wanted to model the growth of neurons (excitable cells) by the transformation of unexcitable cells into excitable ones. Idealized neurons with their two states can handle only logical functions, but construction also requires other operations to acquire and combine elements of which the constructed automaton is composed. Hence, to accomplish construction, von Neumann introduced special stimuli which cause transitions from the unexcitable state to different species of excitable states. Von Neumann's automaton had a set of 29 states.

Von Neumann wanted the space, in which he would carry out his constructions, to have a high degree of regularity. He required functional homogeneity and isotropy [1] and selected a 2-dimensional space. Due to the difficulties of modeling

[1]The space is isotropic if it has the same properties in all directions. In the discrete case, functional isotropy means that each cell is connected to each of its immediate neighbors in the same way.

automata construction in continuous space, he decided to work with discrete space. In summary, his constructing automaton is a 2-dimensional, regular, cellular structure which is functionally homogeneous and isotropic. Von Neumann developed a transition rule for a set of 29 states such that the state of a cell at time $t+1$ depends only on its own state and the states of its four immediate neighbors at time t.

All of von Neumann's questions about automata construction and computation have been answered affirmatively and constructively by means of his 29-state cellular automaton. This automaton was shown to be computation-universal, construction-universal and self-reproducing. In this cellular structure, self-reproduction is a special case of construction, and construction and computation are similar activities. Von Neumann's universal computer-constructor was later simplified by Codd in 1968, who used an 8-state, 5-neighbor cellular automaton.

A cellular model of computation was also considered in the work of Ulam. In [70] he formulated a matrix problem arising out of the cellular model. In [69] and [68], Ulam studied the growth of figures in cellular automata with simple transition rules. He also studied the evolution of successive generations of individuals with simple properties, each generation producing its successor according to a simple but non-linear recursive transformation.

Since that time cellular automata have been used in numerous areas to study different phenomenological aspects of the world. We will discuss various applications of cellular automata in more details in Section 2.5.

2.2 Classical Model

Cellular automata (CA) represent a model of computation that is based on discrete systems. Their computations evolve in discrete time and discrete space. A CA model is defined by four basic components: a discrete CA lattice of elementary automata, a neighborhood template (or simply neighborhood), a set of states that

each automaton can take and, finally, a function that provides rules for local state transitions of automata.

A CA lattice is a d-dimensional array ($d \geq 1$) of elementary CA elements called cells. Every cell is identified by a tuple $\langle i_1, i_2, ..., i_d \rangle$ of integer indices, which represent cell's position on the lattice. We will denote a CA lattice by $\mathcal{L}$ and a cell that belongs to it $x_{\langle i_1, i_2, ..., i_d \rangle} \in \mathcal{L}$.

During CA computation (evolution), each cell can communicate with certain other cells. For any given cell, the set of cells, with which it directly communicates during CA computation, is defined by the neighborhood template. Usually the neighborhood is specified by its radius r. This means that any cell y, that lies at a distance of not more than r cells from cell x, is a neighbor of x. Also, a neighborhood can be defined by a specific template. For example, $u(x_{ij}) = (x_{i-1,j}, x_{i+1,j}, x_{i,j-1}, x_{i,j+1})$. This is a template for the neighborhood in a 2-dimensional cellular automaton. The neighbors in a template can be indexed, such that $u(x) = (y_1, ..., y_k)$, where k denotes the size of a neighborhood.

At any given time of computation, each CA cell is in some state. A non-empty finite set of cell states is denoted by Q and its cardinality by $\mid Q \mid = q$. The state of cell x at time t will be denoted by $x(t)$.

An evolution of a cellular automaton consists in transitions of cell states in discrete time steps at each site of a CA lattice. The rules for local transitions are given by a local transition function f. Each cell x computes its next state, say at time $t+1$, according to this function, which is a function of the current state of cell x as well as states of x's neighbors:

$$x(t+1) = f(u(x(t))) = f(y_1(t), ..., y_k(t)),$$

where $x(t+1)$ is a state of cell x at time $t+1$ and $u(x(t)) = (y_1(t), ..., y_k(t))$ are states of x's neighbors, which includes $x(t)$ as well. $u(x(t))$ is also called the *neighborhood*

configuration of cell x at time t. Thus, the local transition function is a mapping $f : Q^k \rightarrow Q$. It can be described by a formula or table, or any other suitable method. Cells are assumed to change their states synchronously at each time step. Description of cell states on the entire CA lattice at a given time step is called a *cellular automaton configuration.*

Thus cellular automata can be defined by a tuple $\langle \mathcal{L}, Q, u, f \rangle$, where $\mathcal{L}$, Q, u and f define the CA lattice, the set of cell states, the neighborhood template and the local transition function respectively, as discussed above.

The systematic study of cellular automata was initialized by Wolfram. He studied the relationships between CA and different dynamical systems and suggested a classification of CA behavior in this context. According to [77] there are four classes of cellular automata, whose behavior can be compared with the similar behavior of the dynamic systems (given in parenthesis):

1. *Class I:* Automata that evolve to a unique, homogeneous state after a finite number of global cell state transitions (*limit points).*

2. *Class II:* Automata which evolve to a set of simple separated stable or periodic structures (*limit cycles).*

3. *Class III:* Automata whose evolution yields chaotic aperiodic patterns (*chaotic behavior of the kind associated with strange attractors).*

4. *Class IV:* Automata that evolve to complex patterns with propagative localized structures, sometimes long-lived (*very long transients with no apparent analog in continuous dynamic systems).*

Empirical methods are the only feasible means by which any classification of cellular automata can be performed; as it was shown in [21] and [66] it is impossible to decide to which class a given cellular automaton belongs. Classification of CA is

done by means of empirical observations of CA evolution (space-time patterns) in quiescent background and constant-size windows of observability (*circular CAs* [2]).

In [21], a scheme, very similar to Wolfram's but more formal, was suggested. Cellular automata are classified in this study according to the evolution of *finite configurations* [3] even if the computations on all possible configurations are of interest. In this scheme, four classes are defined as a hierarchy: each class contains all lower classes as subsets. It has been proved that for the first three classes this classification coincides with Wolfram's on all linear totalistic [4] cellular automata with two states and the radius of a neighborhood equal to two.

Other classification schemes include classification according to mean field approximations [31], [30], two classification schemes for 3-dimension semitotalistic cellular automata by Bays [8] and dynamic systems classification by Gilman [29], [28].

2.3 Various Models of Cellular Automata.

The model of cellular automata described in the previous subsection is the most basic one. There are also different modifications of the above model. They can include asynchrony of cell state transitions, dependence of cell state transitions on the neighborhood configurations at several earlier steps, the probabilistic or fuzzy nature of a transition function, delayed transitions and varying neighborhood templates. We will briefly present these possible modifications of the classical CA model, mostly after [1]:

[2] A circular CA is a cellular automata in which the leftmost and rightmost cells are the neighbors of each other.

[3] A configuration with all but finitely many cells in the stable state s is called a *finite configuration* with respect to s, where a state s is stable if the condition $f(s, s, ..., s) = s$ holds and f is a transition function.

[4] A cellular automata is called totalistic if its transition function has a form $f(x_1, ..., x_k) = \phi(x_1 + ... + x_k)$, where $x_i \in Q, i \in \{1, ..., k\}$ and ϕ is some function.

1. **Non-uniform Cellular Automata.**

   In a non-uniform cellular automaton, different cells may use different transition functions to determine their next state. The model's complexity increases in this case, so a natural question arises whether the non-uniformity adds any additional computational power to CA. As was indicated in [62], non-uniformity brings some difference to very simple cellular structures. For example, 2-state, 5-neighbor uniform CA are not computationally universal, whereas non-uniform CA are. However, for most uniform cellular spaces, universality can be attained, thus non-uniformity does not increase the power of the CA model in this respect. Actually, it is easy to simulate a non-uniform cellular automaton by a uniform one, encoding different transition rules as one big rule. Nevertheless, experimental results in [62] demonstrate that non-uniformity may reduce connectivity requirements, namely smaller neighborhoods can be used to perform more complex tasks.

2. **Non-stationary Cellular Automata.**

   Non-stationarity in CA model means that a transition function varies with time. Cells of a non-stationary cellular automaton transit to their next state according to the rule $f(u(x(t)), t)$, where function f depends not only on neighborhood configuration, but also on time. The transition function is a mapping $f : Q^k \times T \rightarrow Q$, where T is a set of time steps.

3. **Asynchronous Cellular Automata.**

   In this model of cellular automata, cells make their transitions to the next state asynchronously. We can define two subclasses of the asynchronous cellular automata class $\mathcal{A}_{\mathrm{I}}$ and $\mathcal{A}_{\mathrm{II}}$. A cell of a cellular automaton belonging to class $\mathcal{A}_{\mathrm{I}}$ calculates its next state depending on the neighborhood configuration at time t, makes a transition to this state and remains in it for some number of time steps, calculated according to the function $\xi(u(x(t)))$. In the class $\mathcal{A}_{\mathrm{II}}$, every cell calculates its next state at time t, but makes a transition to it only at time step $t + \delta(u(x(t)))$. Both functions ξ and δ are mappings $Q^k \rightarrow D$,

where D is a set of delays, $D = \{0, 1, 2, ..., h\}$, with h being a maximal delay.

4. **Cellular Automata with Memory.**

   A transition function of a cellular automaton with memory depends not only on the neighborhood configuration at time step t, but on those at some earlier time steps. Transition function has thus the form $f(u(x(t)), u(x(t-1)), ..., u(x(t-\gamma)))$, where γ is a capacity of cell memory.

   Different combinations of the above modifications can be accommodated in one cellular automata model. In addition they can be combined with the modifications presented next. The following is an example of such a combination.

5. **Probabilistic Cellular Automata.**

   Probabilistic cellular automata accommodate both the classical cellular automata theory and the theory of Markov processes locally interacting in discrete time. Let us consider a class of probabilistic asynchronous cellular automata $\mathcal{PA}_{\mathrm{p}}$ and its subclasses. Class $\mathcal{PA}_{\mathrm{p}}$ can be defined by a tuple $\langle \mathcal{L}, \mathrm{Q}, \mathrm{D}, \mathrm{u}, \pi, \delta \rangle$, where $\mathcal{L}$, Q and u are the same as before.

   - For any neighborhood configuration $w \in Q^k$ and every cell state $a \in Q$ there exists a conditional probability $\pi(w, a)$ of a local transition of a cell x into a state a when its neighborhood $u(x)$ currently has configuration w. Thus, π is a probability distribution function of cell state transitions: $\pi : Q^k \times Q \to [0, 1]$, where [0,1] is the unit real interval, $\forall w \in Q^k, \forall a \in Q, \pi(w, a) \geq 0, \sum_{a \in Q} \pi(w, a) = 1$.
   - This model is asynchronous and belongs to the class $\mathcal{A}_{\Pi}$. This means that at a time step t, a cell x calculates its next state, but it doesn't necessarily make a transition to it at the time step $t + 1$. It may delay its transition for some number of time steps. D is a set of delays of cell-state transitions as explained previously. Furthermore, the actual delay is determined in a probabilistic fashion. Cell x implements a transition after delay b with conditional probability $\delta(u(x(t)), b)$. Thus δ is a probability

distribution function of transition delays: $\delta : Q^k \times D \rightarrow [0,1], \forall w \in Q^k, \forall b \in D, \delta(w,b) \geq 0, \sum_{b \in D} \delta(w,b) = 1$.

Class $\mathcal{PA}_p$ forms a hierarchy of classes (including $\mathcal{PA}_p$ itself), presented in Table 2.1, where f and d are the deterministic functions for cell-state transitions and delays of transitions respectively. In this table $\mathcal{PS}$ stands for probabilistic synchronous, $\mathcal{PA}$ - probabilistic asynchronous with deterministic delays, $\mathcal{DA}_p$ - deterministic asynchronous with probabilistic delays, and $\mathcal{PA}_p$ - probabilistic asynchronous with probabilistic delays.

Table 2.1: Classes of Probabilistic Cellular Automata Hierarchy

Subclass	Characterization
$\mathcal{PS}$	$\pi : Q^k \times Q \rightarrow [0,1], \forall w \in Q^k, \forall a \in Q, \pi(w,a) \geq 0, \sum_{a \in Q} \pi(w,a) = 1$
$\mathcal{PA}$	$\pi : Q^k \times Q \rightarrow [0,1], \forall w \in Q^k, \forall a \in Q, \pi(w,a) \geq 0, \sum_{a \in Q} \pi(w,a) = 1$
	$d : Q^k \rightarrow D$
$\mathcal{DA}_p$	$f : Q^k \rightarrow Q$
	$\delta : Q^k \times Q \rightarrow [0,1], \forall w \in Q^k, \forall a \in Q, \delta(w,a) \geq 0, \sum_{a \in Q} \delta(w,a) = 1$
$\mathcal{PA}_p$	$\pi : Q^k \times Q \rightarrow [0,1], \forall w \in Q^k, \forall a \in Q, \pi(w,a) \geq 0, \sum_{a \in Q} \pi(w,a) = 1$
	$\delta : Q^k \times D \rightarrow [0,1], \forall w \in Q^k, \forall b \in D, \delta(w,b) \geq 0, \sum_{b \in D} \delta(w,b) = 1$

The definitions presented in Table 2.1 serve as examples of how the probabilistic features can be introduced into different parts of a cellular automata model- in our case, a cell state transition function and/or a transition delay mode. This flexibility of the model's definition may help cellular automata to adequately reflect different kinds of uncertainty present in the phenomena which cellular automata are designed to model.

6. **Fuzzy Cellular Automata.**

Fuzzy theory was introduced to automata theory more than twenty years ago in [75]. Other studies of fuzzy automata were conducted in [45], [74]. The work in [51] and [59] dealt with fuzzy automata built on the basis of neuron-like elements. Let us define a class of fuzzy asynchronous cellular automata as

a tuple $\langle \mathcal{L}, Q, D, u, \mu \rangle$, where Q is a finite non-empty set of fuzzy cell states. Let Q^k be a set of all the possible fuzzy configurations of a neighborhood of the size k. Then, $\mu : Q^k \times Q \times D \rightarrow [0,1]$ is such a mapping that for $b \in D, \mu(u(x(t)), x(t+b))$ is a grade of transition of cell x from state $x(t)$ at time step t to state $x(t+b)$ at time step $t+b$ when the neighborhood configuration of cell x at time t is $u(x(t))$. Furthermore, cell x stays in state $x(t)$ in the time interval $[t, t+b-1]$.

We will discuss fuzzy cellular automata in more details in Chapter 6.

7. **Hierarchical Cellular Automata.**

   A hierarchical cellular automaton is a finite sequence of finite 1-dimensional deterministic cellular automata $\mathcal{U}_0$, $\mathcal{U}_1$,...,$\mathcal{U}_p$, where cellular automaton $\mathcal{U}_i$ is controlled by cellular automaton $\mathcal{U}_{i+1}$ for $i = 0, 1, ..., p-1$ and an automaton $\mathcal{U}_p$ has no supervisor and evolves in the usual way. Let l_{i+1} be the size of the cellular automaton $\mathcal{U}_{i+1}$. Then the configuration of the cellular automaton $\mathcal{U}_{i+1}$, $c_{i+1} \in Q^{l_{i+1}}$, defines the transition rule for the automaton $\mathcal{U}_i$. Thus, $l_{i+1} = q^{k_i}$ where $q = |Q|$, and k_i is the size of the neighborhood of the automaton $\mathcal{U}_i$. All possible neighborhood configurations of the automaton $\mathcal{U}_i$ are arranged in the lexicographic list and each of them is identified with the unique integer from the set $\{1, ..., q^{k_i}\}$ using an isomorphism $\nu : Q^{k_{i+1}} \leftrightarrow \{1, ..., q^{k_i}\}$. In other words, an isomorphic function ν assigns a unique index to each neighborhood configuration of the automaton $\mathcal{U}_i$. If the neighborhood state configuration of cell x of the automaton $\mathcal{U}_i$ is $u_i(x)$ then this cell makes a transition to a state $c_{i+1,j}$, where $j = \nu(u_i(x)) \in \{1, ..., l_{i+1}\}$.

8. **Structurally Dynamic Cellular Automata.**

   Structurally dynamic cellular automata were suggested in [35] introducing the idea that the lattice structures can be dynamically coupled to the cell states. Every cell in such an automaton evolves accordingly to the rule $x(t+1) = f(v(u(x(t-1))))$, i.e. the next state of a cell is determined by the state of the neighborhood which in its turn is calculated from the neighborhood

configuration at time $t-1$. Representatives of this class can also accommodate probabilistic, fuzzy and other features.

As one can see, cellular automata model of computation allows a rich variety of features which can help it to adapt to specific computational needs.

2.4 Cellular Automata as a Universal Model of Computation.

When we say that cellular automata are capable of universal computation, we mean that their computational power is equivalent to that of a universal Turing machine [34]. The first cellular automaton, capable of universal computation was described by von Neumann. Von Neumann's universal computer-constructor was simplified by Codd in [19] who gave a description of a 2-dimensional, 8-state, 5-neighbor universal cellular automaton. The cellular automaton, known as a "game of life", which involves 2-dimensional, 2-state and 9-neighbor cellular space, was proven to support universal computation with finite initial configurations [9]. In [5], 2-dimensional 2-state and 3-state automata, both 5-neighbor, were described and proved to support universal computation with infinite and finite initial configurations respectively. One dimensional cellular automata have also been shown to support universal computation [64]. We will briefly illustrate this case.

One-dimensional cellular automata are capable of universal computation if for each Turing machine M, we can construct a 1-dimensional cellular automaton whose transition rule depends on M and which simulates M in the following sense. An input string w recorded on a tape of a Turing machine M, can be encoded into an initial configuration w' of the cellular automaton. Computation of the cellular automaton starting from the initial configuration w' encodes step by step the computation of M started with w on the tape. The program of M can be encoded

into the cellular automaton transition rule. We will briefly show how 1-dimensional cellular automata can simulate the computation of a Turing machine.

We will refer to a Turing machine with a set B of n states and an alphabet A of cardinality m as (m, n) Turing machine. A program of a Turing machine can be specified by a state table of size $m \times n$. Each entry of this table $(t_{i,j})$ has a form $a_i' X q_j'$: if the head of the Turing machine reads a symbol $a_i \in A$ on a tape and the Turing machine is currently in a state $q_j \in B$, then the Turing machine writes a symbol $a_i' \in A$ on a tape, the head moves in direction $X \in \{Right, Left\}$ and Turing machine transits to a state $q_j' \in B$.

Theorem [64]. An arbitrary (m, n) Turing machine M can be simulated by a 1-dimensional, $(m + 2n)$-state, 3-neighbor cellular automaton CA with the neighborhood template $u(x_i) = (x_{i-1}, x_i, x_{i+1})$ in at most two times real time.

The idea of the proof of this theorem is the following. The m states of CA correspond to the symbols of M's alphabet. Each square on a Turing machine tape is encoded by a CA cell, which takes a state corresponding to a symbol recorded on that square. Each state $q \in B$ of M has two corresponding CA states (q and q_L). An initial state of M is encoded by a cell immediately to the left of the cell encoding the first symbol of the input string on M's tape. Similarly, at further steps of the computation, the state of M is always encoded by a cell immediately to the left to the one encoding the symbol read by M at this step. Let us denote a cell currently encoding M's state - s, a cell to the left of it - p and a cell to the right - h. The CA transition function leaves the states of all cells unchanged except the states of the cells p, h and s. Assume that current CA configuration is $...s_0 s_1 q s_2 s_3...$, i.e. M is in a state q and M's head is reading the symbol s_2. Assume that according to M's state table, M should write symbol s_2' on the tape, move the head right and transit to a state q'. Then the CA will make a transition:

$$...s_0 s_1 q s_2 s_3... \text{ time step } t$$
$$...s_0 s_1 s_2' q' s_3... \text{ time step } t + 1$$

In the case of the left move of the head, the CA will make two transitions:

$$...s_0s_1qs_2s_3... \text{ time step } t$$
$$...s_0s_1q'_Ls'_2s_3... \text{ time step } t+1$$
$$...s_0q's_1s'_2s_3... \text{ time step } t+2$$

The CA transition function is derived from M's state table to enable the simulation of M's computation as described above.

The simulation of a Turing machine computation by cellular automata can be done in many ways. For example there is 1-dimensional, 4-neighbor $(m+n)$-state cellular automaton with the neighborhood template $u(x_i) = (x_{i-1}, x_i, x_{i+1}, x_{i+2})$ which simulates (m,n) Turing machine in real time [64].

If the simulation of a Turing machine by cellular automata is done for a universal Turing machine, one can obtain a universal cellular automata. As indicated in [20], a cellular automaton CA_u is strongly universal if any cellular automaton CA with local rule $f : Q^k \rightarrow Q$ and any initial configuration w over Q can be encoded into an initial configuration of CA_u so that each cell of the simulated CA is encoded as a block of cells of CA_u, where each of these blocks also encodes the local rule f. In [3] such an automaton with 14 states was described.

From a theoretical point of view, it is indeed interesting and important that cellular automata are capable of universal computation. But this property seems to be less relevant for a researcher who aims for applications of cellular automata to the practical problems arising from natural and man-made systems. As we could see earlier, universal computation in cellular automata arises only for some specifically designed initial configurations. In practice there are efficient ways to model natural processes with cellular automata other than to program cellular automata designed for universal computation, to perform a specific practical task. Thus, the expediency and efficiency of cellular applications to various specific problems should be assessed despite of the fact of cellular automata's ability to perform universal computation.

2.5 Applications of Cellular Automata.

The most popular approach to modeling of various natural systems involves the usage of differential equations. A theory of differential equations has been developing for many years and has become a very important area of mathematics that provides a framework for systems' studying in many other disciplines. But alternative approaches to modeling nature are also considered, amongst which cellular automata offer many interesting features.

There are numerous complex natural phenomena whose macroscopic behavior is still not well studied despite of the fact that the microscopic laws underlying their nature are known and are often quite simple. A simple example of such a phenomenon can be the formation of a snowflake: though the laws of thermodynamics, on the micro level, are well understood the prediction of a snowflake's pattern is an extremely difficult problem. From this perspective, cellular automata computations exhibit a similar characteristic - generation of complex behavior by the cooperative effects of many simple components. This feature of cellular automata as computational devices provides a tool by which methodologies and results from computational theory can be directly applied to such sciences as physics, biology, sociology, medicine etc., and vice versa. For example, classification of cellular automata by Wolfram [77] already establishes connections between properties of a cellular automata computational model and those of natural or man-made dynamic systems. Cellular automata have been extensively studied and used as models of a variety of physical systems. As illustrated in [71], one may distinguish three approaches to using cellular automata to simulate physics.

1. *Cellular automata as computational tools.* Hardware implementations of cellular automata can be viewed as another alternative to special purpose machines and general purpose computers. They can be seen as general-purpose machines for discrete problems of local interactions. The validity of the simulation by cellular automata of space-discrete systems (lattices) depend on the effect of discretizing time

and state variables of the simulated system. Cellular automata find their most natural applications in those areas of physics where a discretization of space is a feature of a physical system itself, rather than an inevitable feature of numerical simulations. An example of such a physical system can be crystals with their lattice-vibration independent properties. Cellular automata are equally naturally applicable to simulate those physical systems for which spatial discretization has been made an integral part of an established theoretical model. An example can be a lattice-gas molecular dynamics. In these cases, cellular automata attempt to match simulated system's structure and topology. The analogy is particularly apparent for dynamic models. In fact, many numerical results in the study of percolation, nucleation, condensation, coagulation and transport properties in molecular dynamics are obtained by simulating time evolution of systems [71]. Many of these simulations were conducted by the means of cellular automata or their generalizations. For static models, on the other hand, the correspondence is less immediate because cellular automata deal with "initial condition" problems, and in the classical mechanical theories, there is no natural microscopic dynamics. But cellular automata can also be useful if they reproduce the successive steps of some iterative method, where the automaton's time plays a role of an iteration index rather than the physical time.

2. *Cellular automata as fully discrete dynamic systems.* A given cellular automata defines its own discrete universe. It turns out that many of these cellular automata universes have properties that are most often seen in the theoretical physics. Amongst them - the appearance of complex phenomena and large-scale correlations resulting from very simple short range interactions. This approach demonstrates a power and generality of concepts which, though conceived for a physical problem, can be applied and defined in the contexts totally unrelated to physics. Cellular automata indicate that these concepts have a broader scope because they are ultimately based on principles of computational and informational nature. Thus, discrete dynamic systems attempt not to simulate specific physical phenomena, but rather to embody general physical ideas.

3. *Cellular automata as original models for actual physical phenomena.* This is the most ambitious approach in using cellular automata to simulate physics, which suggests that cellular automata can compete with existing continuum models. But so far nobody came across a cellular automaton that can pretend to be an original model of a physical system or phenomenon.

These approaches have an important common feature. They provide a third alternative to models that are *solvable exactly* (by analytical means) but are very stylized, and models that are more realistic but can be *solved approximately* only (by numerical means). Cellular automata have enough expressive power to represent phenomena of arbitrary complexity and at the same time they can be simulated exactly by hardware cellular automata devices. This introduces a third class - *exactly computable* models. There is no attempt to solve any given equation, cellular automata is not involved in any kind of numerical processing, they perform only simple space-dependent logical decisions. This third class aims at digital non-numerical simulations of physical phenomena.

Cellular automata are also used as models of biological systems and provide a mathematical basis for investigation of complex behavior of living organisms. As discussed in Section 2.1, the cellular automata model was originally invented in order to study self-reproduction, a property often considered as being definitive for living systems. A majority of complex biological systems cannot be precisely quantified [25], and detailed models are extremely difficult to obtain. Often such models lead to formidable numerical problems. One technique accepted in biology to overcome this problem is to mimic the physical laws by a series of simple rules that can be computed quickly and in parallel. This idea immediately suggests a utilization of cellular automata. The most valuable feature that cellular automata offer to biology and medicine researchers is the ability to model spatial and temporal pattern formations. The most popular cellular automata models used in biology are deterministic cellular automata, "lattice gas" models of cellular automata and "solidification" models.

Deterministic cellular automata are often treated as tools to mimic partial deferential equations. In this context they are often applied to model waves in excitable and oscillatory media (nerve and muscle cells [61], cardiac function [60], chemical reactions) as well as predator-prey models [58] and spatial pattern formation [65].

Lattice gas models are the systems that consist of a discrete spatial grid on which particles move and interact. These systems are usually driven by random events. This kind of cellular automata can be used to model self-organization of ants' traits, fibroblast aggregation and topographic neural maps [25].

A "solidification" model is similar to a lattice gas model, except that there are some special, "bound" states, and once a particle is in a bound state, it can never move or disappear. This type of model is useful in modeling phase-transitions and precipitation processes (e.g. formation of vascular networks [16], growth of immotile colonies of bacteria under the conditions of nutrient limitations [42]).

In [25], cellular automata are viewed as an important and useful tool in biological research, but considered to be still insufficient for derivation of some absolutely general results. The reason for this is that it is in very rare cases that some particular behavior of cellular automata can be proven. Normally, the only available technique to verify some property is to run a simulation, which is not a good strategy when some generalized conclusions are expected.

Cellular automata appear to be useful for modeling of various socio - environmental systems. Often, a macroscopic approach to a modeling task like this is not sufficient, since local spatial and organizational details are extremely important in order to comprehend the overall dynamics of such systems. Cellular automata offer a natural way to model basic elements of the systems' space (or basic components) by cellular automata cells and reproduce their interactive behavior in time through a cellular automata computation. In [24], cellular automata were applied to a problem of forecasting socio-economic and environmental effects of climate change on an island. Fuzzy cellular automata was used to predict wind driven wild land fires in

[46]. Rules and strategies of technological competition within social networks were studied with the help of cellular automata in [22]. Economy, as a dynamic system, was modeled by means of cellular automata in [4] and [50]. Urban planning benefited from the use of cellular automata as well [7], [52], [76].

In all the cases of cellular automata applications discussed above, the underlying rules of local interactions between the system's components (and thus cellular automata cells modeling them) were known but macroscopic (global) behavior of a system needed to be studied. However, knowledge of local laws is not always available. In this case various techniques from machine learning can be used to identify these local rules. This subject will be discussed in more details in Section 4.3.

As it was pointed out previously, cellular automata seem to be best applied to systems which possess an intrinsic property of a discretized space and when a correspondence between system's components and a cellular automata structure is natural and apparent. But is this a limit of cellular automata applications? The ability of cellular automata to represent knowledge of a different kind still has to be investigated and the efficiency of such representation be evaluated and compared with other techniques. If this evaluation turns out to be positive, cellular automata with their characteristics of fast parallel computational devices can offer a powerful alternative to existing digital computers and computational models currently used in artificial intelligence.

Chapter 3

Fuzzy Theory.

3.1 Motivations of Fuzzy Set Paradigm.

The traditional point of view of the earlier science expressed a belief that uncertainty is absolutely undesirable for science and it should be avoided by all means. According to the alternative (modern) view, uncertainty is not only a reality that science can not avoid, but rather it is one that can have a great utility.

Until recently, there were two approaches used in science: analytic methods based on the calculus and statistical methods based on the probability theory. The former methods are applicable to problems that involve a rather small number of variables that are related to each other in some predictable way. Statistical methods, on the contrary, require a large number of variables and a high degree of randomness. These two types of methods cover two extremes of problems usually encountered in practice. They were referred to by Warren Weaver in [73] as problems of *organized simplicity* and *disorganized complexity*. He also pointed out that these problems constitute only a tiny fraction of all systems problems. The majority of problems exist between these two extremes; they often include non-linear systems with large numbers of components involved in rich interactions. These interactions are usually

nondeterministic but *not* as a result of randomness that could yield meaningful statistical averages. These problems are called *problems of organized complexity.* Such problems often deal with modeling of different systems or phenomena, be it natural or man-made. These models are used to make predictions or retrodictions about the systems, control some phenomena, etc. In constructing a model, one would almost always face the presence of uncertainty. But it is not always an undesirable obstacle; in general, allowing more uncertainty tends to reduce complexity and increase credibility of the resulting model [37]. The challenge is then to estimate an optimal level of allowable uncertainty.

The emergence of the new concept of a fuzzy set in [78] reflected the recognition of the opinion that uncertainty is an important issue and a useful feature in modeling. It was motivated by the necessity to deal with the gap which existed between mathematical models and their empirical interpretations in most of real-life phenomena. This gap became especially apparent in such areas as biology, medicine, cognitive and the social sciences. As was pointed in [10]: "It is a paradox, whose importance familiarity fails to diminish, that the most highly developed and useful scientific theories are ostensibly expressed in terms of objects never encountered in experience."

There are many features that make the paradigm of fuzzy theory very useful, amongst which are the following, as indicated in [37]:

1. Fuzzy sets allow us to express irreducible observation and measurement uncertainties and make them intrinsic to empirical data. These uncertainties are processed together with fuzzy data, and the results obtained are more meaningful from a practical point of view than those obtained by processing standard data.

2. As was indicated previously, uncertainty introduces a tool for reducing the complexity and computational cost of a problem.

3. Fuzzy sets have a capability of expressing meanings of linguistic concepts. This additional expressive power allows us to deal with problems that require the use of natural language.

4. Fuzzy theory can better capture human common-sense reasoning, decision making and other aspects of human cognition.

The relationship between probability theory and fuzzy set theory is a very controversial issue in uncertainty modeling and information sciences. Fuzzy sets and probability measure are considered to be distinct [23], but their opposing points of view on uncertainty modeling can be reconciled, to some degree, by means of possibility theory. Both possibility theory and probability theory are special branches of evidence theory, which is based on basic two measures - belief and plausibility. Measures employed in possibility theory - possibility and necessity - are special cases of belief and plausibility, and are defined only for the families of nested sets. Probability theory coincides with evidence theory only for objects for which belief and plausibility measures are equal. Probability measure is defined only for objects which are singletons.

Possibility, necessity and probability measures do not overlap with one another [37], except for one very special case: one element of the universal set is assigned the value of all three measures equal to 1, with all other elements being assigned the values of all measures equal to 0. This case represents availability of the perfect evidence. There are many interpretations of probability theory as well as possibility theory. Some of them suggest absolutely no connection between two theories but some do. For example, possibility theory may be interpreted in terms of interval-valued probabilities: possibility measure is considered as an upper probability estimate and necessity measure as a lower probability estimate [37], [23], [57]. This point of view opens a bridge between two theories. Since possibility theory can also be defined in terms of fuzzy sets [37], the bridge is also established between probability theory and fuzzy sets theory.

The difference in mathematical properties of the measures used in probability and possibility theories make each of them suitable for modeling certain types of uncertainty [37]. For example, probability theory is an ideal tool for formalizing uncertainty in situations where class frequencies are known or where evidence is based on outcomes of a sufficiently long series of independent random experiments. Possibility theory, on the other hand, is good for formalizing incomplete information expressed in terms of fuzzy propositions. The motivation to study probability-possibility relationships has arisen not only out of desire to compare two points of view on uncertainty, but also from certain practical problems. Examples of such problems are: constructing a membership grade function of a fuzzy set from statistical data, constructing a probability measure from a given possibility measure in the context of decision making or systems modeling, combining probabilistic and possibilistic information in expert systems or transforming probabilities to possibilities to reduce computational complexity. To deal with these problems, various probability-possibility transformations have been suggested in the literature [37], [23], [39], [38].

3.2 Fuzzy Sets and Standard Fuzzy Operations.

A fuzzy set is a class that admits the possibility of the partial membership of objects. Fuzzy sets were first introduced in [78] as sets with boundaries that are not precise. The membership in a fuzzy set is not a matter of affirmation or denial, but a matter of *degree*. A fuzzy set can be defined mathematically by assigning to each possible object in the universal class a value representing its grade of membership in the fuzzy set. This grade represents the degree to which that individual element is similar to the concept represented by the fuzzy set.

Given a set of objects X, any arbitrary fuzzy set $A \in X$ is defined by its membership function $\mu_A : X \rightarrow [0,1]$. For each object $x \in X$, the membership function

indicates the degree of membership of the object x in the fuzzy set A, with 1 representing full membership and 0 - non-membership. Sometimes the identifiers of a fuzzy set and its membership function are not distinguished. A fuzzy set defined in this way is the most basic and common type of fuzzy sets and is often called an *ordinary fuzzy set.* Membership functions can have different forms, which are determined by particular applications. They can be continuous as well as discrete, in which case they can be represented by a table of values or rules.

One of the generalizations of the above definition involves fuzzy sets whose elements are ordinary fuzzy sets. They are known as level 2 fuzzy sets. Their membership functions have the form $\mu_A : \mathcal{F}(X) \rightarrow [0,1]$ where $\mathcal{F}(X)$ denotes the fuzzy power set of X (the set of all ordinary fuzzy sets of X). Level 2 fuzzy sets allow us to deal with situations in which elements in X can not be specified precisely, but only by other fuzzy sets, that express propositions of the form "x is close to r" where r can be defined exactly. Level 2 fuzzy sets can be generalized into higher level fuzzy sets recursively.

Fuzzy sets can also represent linguistic concepts (e.g. low, medium and high temperature). Such sets are often used to define the states of some variables. In this case a variable is called a fuzzy variable and its states are defined by membership functions of the corresponding fuzzy sets. The useful property which fuzzy variables possess is that they enable gradual transition between states and consequently can tolerate to some degree observation and measurement uncertainties. Traditional variables do not have this property. Because of this, fuzzy variables can be more useful in real-life applications, where some level of uncertainty almost always exists.

Given two fuzzy sets, A and B, whose membership functions are denoted by the same identifiers as the corresponding sets, we can define *standard fuzzy set*

operations:

$$\begin{aligned} (A \bigcup B)(x) &= \max(A(x), B(x)) && \text{- standard union} \\ (A \bigcap B)(x) &= \min(A(x), B(x)) && \text{- standard intersection} \\ \neg A(x) &= 1 - A(x) && \text{- standard complement} \end{aligned} \tag{3.1}$$

Fuzzy sets can be formed of other fuzzy sets by application of the above operations.

One can see that if in classical sets membership grades are restricted to 0 and 1 only, then the standard fuzzy operations are identical to the standard set operations. Standard fuzzy operations are not the only possible generalization of classical set operations. Functions that play a role of fuzzy intersections have a general name *t-norms* , and fuzzy unions - *t-conorms* . Standard fuzzy intersection (min operator), applied to any fuzzy sets, produces the largest resulting fuzzy set as compared to those produced by other t-norms. The standard fuzzy union, on the other hand, produces the smallest fuzzy set amongst the results of other t-conorms [37].

One useful property of standard fuzzy operations is their ability to prevent compounding of errors of operands. If e is a maximum error which is present in membership grades $A(x)$ and $B(x)$, then the error of the results of standard fuzzy operations, applied to $A(x)$ and $B(x)$, remains e.

One can represent standard fuzzy operations in a different but equivalent form. Consider the following:

$$\max(a, b) = \frac{1}{2}(a + b + \mid a - b \mid) \tag{3.2}$$

Indeed, if $a > b$, $\mid a - b \mid = a - b$ and $\max(a, b) = \frac{1}{2}(a + b + a - b) = a$. If $a < b$, $\mid a - b \mid = b - a$ and $\max(a, b) = \frac{1}{2}(a + b + b - a) = b$.

Similarly we can represent

$$\min(a, b) = \frac{1}{2}(a + b - \mid a - b \mid) \tag{3.3}$$

Several fuzzy sets (two or more) can be combined to produce a new single fuzzy set. This can be done by means of aggregation operations. Any aggregation operation on n fuzzy sets ($n \geq 2$) is defined by a function of a form:

$$h : [0,1]^n \rightarrow [0,1] \tag{3.4}$$

When function h is applied to fuzzy sets $A_1, A_2, ..., A_n$, which are defined on object class X, then an aggregate fuzzy set A, produced by h, can be defined as:

$$A(x) = h(A_1(x), A_2(x), ..., A_n(x)) \tag{3.5}$$

Function h operates on the membership grades of the fuzzy sets $A_1, A_2, ..., A_n$.

Intuitively, a meaningful aggregation function h should satisfy at least the following three axioms:

Axiom h1. $h(0,0,...,0) = 0$ and $h(1,1,...1) = 1$ (*boundary conditions).*

Axiom h2. For any pair $\langle a_1, a_2, ..., a_n \rangle$ and $\langle b_1, b_2, ..., b_n \rangle$ of n-tuples such that $a_i, b_i \in [0,1]$, $i \in \{1,2,...,n\}$, if $a_i \leq b_i$ for all $i \in \{1,2,...,n\}$, then $h(a_1, a_2, ..., a_n) \leq h(b_1, b_2, ..., b_n)$; that is h is monotonic increasing in all its arguments.

Axiom h3. h is a continuous function.

It is easy to see that standard fuzzy intersection and union satisfy axioms h1-h3 and can be used as aggregation operations. Due to the associativity of min and max operators, their definitions can be extended to any finite number of arguments. Consider operations min(S) and max(S), where $S = \{s_1, s_2, ..., s_n\}$. Then, $\forall s_i \in S$,

$$\max(S) = \max(\max(S \setminus s_i), s_i)$$
$$\min(S) = \min(\min(S \setminus s_i), s_i)$$

and $\max(S \setminus s_i)$ and $\min(S \setminus s_i)$ can be calculated recursively.

Chapter 4

Learning Process.

4.1 Learning Paradigms.

There are a lot of notions associated with the term "learning". In our study, we will mean by learning a process by which free parameters of an artificial system (computer program) that learns are updated as a result of the interaction with the environment whose behavior the artificial system attempts to model. A learning system can be required to perform different tasks, such as function approximation, prediction, association, pattern classification, control, acquiring rules for expert systems, integral reasoning etc. One can identify four major paradigms being investigated in today's machine learning domain [14]:

- **Inductive Learning.** This method of learning induces a general concept description from a sequence of instances of the concept and sometimes known counterexamples of the concept.
- **Analytic Learning.** This paradigm of learning is based on analytic learning from few exemplars (often a single one) plus a rich underlying domain theory. The methods included in this paradigm are deductive. They utilize past

problem solving experience (the exemplars) to guide which deductive chains to perform when new problems have to be solved, or to formulate new search rules that enable more efficient application of domain knowledge. Analytic methods focus on improving the efficiency of a system without sacrificing the accuracy of generality. Representatives of the analytic learning are methods of explanation based learning, multi-level chunking, iterative macro-operators and derivational analogy.

- **Genetic Algorithms.** Genetic algorithms have been inspired by a direct analogy to mutations in biological reproduction (cross-overs, point mutations, etc.) and Darwinian natural selection (survival of the fittest in each ecological niche). Variants of concept description correspond to individuals of the species and recombinations of these concepts are tested against an objective function to determine which to preserve in the gene pool. In principle genetic algorithms encode a parallel search through concept space, with each process attempting coarse-grain hill climbing.

- **Connectionist Learning.** Connectionist learning systems are parallel distributed systems, of which artificial neural networks are the main representatives. In the process of learning, connectionist systems are presented with training sets of representative instances of some concept (possibly with some noise) and they learn to recognize these and other instances of the concept. Learning consists in readjusting weights - free parameters of a system - via different learning algorithms.

In very simplified and generalized terms, one may say the following [14]. Connectionist approaches are superior for learning in unstructured continuous domains, if a large number of training examples is available. At the opposite end of the spectrum, analytic methods are best for well-structured knowledge-rich domains that require deep reasoning and multi-step inference even if few training examples are present. Inductive and genetic methods are useful for the applications in the center between

those two extremes. Of coarse, there are many tasks that may be approached by more than one method and there are complex tasks where multiple forms of learning should coexist.

4.2 Connectionist Systems: Learning Algorithms and Training Paradigms.

The concepts that constitute the basis of the learning algorithms for connectionist systems are of the great interest for our study. They can potentially be applied to the learning with cellular automata, which can be considered as a kind of connectionist systems. In this subsection we will focus our attention on the most important approaches used in the connectionist learning.

In this section we will use the term "artificial learning" system (or simply "learning system") referring to a connectionist learning system.

A type of learning is determined by the manner in which the free parameters of an artificial system are updated. A prescribed set of well-defined rules by which this updating is driven is called a learning algorithm. We will discuss four main types of learning: error-correction learning, Hebbian learning, competitive learning and Boltzmann learning. Another factor that determines the learning process is the manner in which an artificial system interacts with the environment and is called a training paradigm. We will mention three training paradigms: supervised learning, reinforcement learning and self-organized (unsupervised) learning [32].

4.2.1 Types of Learning.

- **Error - Correction Learning.**

  Let $d(\ell)$ denote some desired response (or target response) of an artificial

system at time ℓ. Let the corresponding value of the actual response of the artificial system be denoted by $y(\ell)$. This response $y(\ell)$ is produced as a result of presentation of an input (stimulus) vector $\mathbf{x}(\ell)$ to the input of the artificial system. The input $\mathbf{x}(\ell)$ and desired response $d(\ell)$ constitute a particular example presented to the artificial system at time ℓ, and such examples are generated by an environment whose behavior one wants to model with the artificial system. Typically, the actual response $y(\ell)$ is different from the desired one and one can define the error between the target response and the actual one:

$$e(\ell) = y(\ell) - d(\ell) \tag{4.1}$$

The goal of the error correction learning is to minimize a *cost function* which is based on the error (4.1). Once a cost function is selected, an error - correction learning becomes an optimization problem with respect to free parameters of the artificial system. A popular choice of the cost function is *mean-square-error:*

$$I = E[\frac{1}{2}e^2(\ell)] \tag{4.2}$$

where E is the statistical expectation operator. The difficulty of this optimization procedure lies in the fact, that statistical characteristics of the underlying process are not known. To overcome this difficulty we aim for an approximate solution of the optimization problem by using an instantaneous value of (4.2) as a cost function:

$$\varepsilon(\ell) = \frac{1}{2}e^2(\ell) \tag{4.3}$$

A plot of the cost function versus free parameters of an artificial system is referred to as an *error surface.* Thus, the objective of the error-correction learning is to start from an arbitrary point on the error surface (determined by the initial values of free parameters) and move toward a minimum in a step-by-step fashion.

- **Hebbian Learning.**

Hebbian learning is based on Hebb's postulate of learning and is so named in

honor of the neurosychologist Hebb. This postulate says[32]: "When an axon of a cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased."

This postulate was stated in a neurobiological context. The notion of the synapse can be extended to the transmission site through which information-baring signals are transmitted in time and space. The Hebbian synapse is a synapse whose strength is increased if the activities on both sides of this synapse are positively correlated and is weakened if these activities are not correlated or negatively correlated. The mechanism, that is responsible for the change in strength of the Hebbian synapse, depends on the exact time of occurrence of the pre- and postsynaptic activities. This mechanism is highly local since the activities used to determine the change in the synapse's strength are only those that are local (adjacent) to the synapse. The occurrence of the change in a Hebbian synapse always depends on the activity levels on both sides of the synapse.

- **Competitive Learning.**

  In an artificial learning system based on competitive learning, the output elements compete among themselves to be activated, with the result that only one output element, or one element per group, is active at any one time. The output elements that win the competition are called *winner-takes-all elements.*

  Competitive learning forms a basis for *self-organizing feature maps.* In such systems, elements are placed at the nodes of a lattice that is usually one- or two-dimensional; higher dimensions are also possible but not common. The elements become selectively tuned to various input patterns or classes of input patterns in the course of a competitive learning process. The locations of the elements so tuned tend to become ordered with respect to each other in such a way that a meaningful coordinate system for different input features is created over the lattice. A self-organizing feature map is therefore characterized by

the formation of a *topographic map* of the input patterns, in which the spatial locations of the elements in the lattice correspond to intrinsic features of the input patterns.

- **Boltzmann Learning.**

  The Boltzmann learning rule is a stochastic procedure derived from information-theoretic and thermodynamic considerations. In a Boltzmann machine, the elements - neurons - constitute a recursive structure and operate in binary manner: they can be either in the state "on" denoted by "+1" or in state "off" denoted by "-1". The machine is characterized by an energy function:

$$E = -\frac{1}{2}\sum_{i}\sum_{j,j\neq i} w_{ij}s_j s_i \tag{4.4}$$

where s_i is the state of neuron i, and w_{ij} is a synaptic weight connecting neuron i to neuron j. This machine operates by choosing a neuron at random - say, neuron j - at some step of the learning process and flipping the state of neuron j from state s_j to state $(-s_j)$ at some temperature T with probability

$$W(s_j \rightarrow -s_j) = \frac{1}{1+\exp(-\frac{\triangle E_j}{T})} \tag{4.5}$$

where $\triangle E_j$ is the energy change resulting from such a flip. Note, that T is not a physical temperature but a pseudotemperature. The neurons of the Boltzmann machine are partitioned into two groups: visible and hidden. Visible are those neurons that provide an interface between the machine and the environment. Hidden neurons always operate freely. The Boltzmann machine has two modes of operation: *clamped condition* , in which all the visible neurons are clamped onto specific states determined by the environment; *free-running condition* , in which all neurons (visible and hidden) are allowed to operate freely.

Let ρ_{ji}^{+} denote the correlation between the states of neurons i and j conditional on the network being in its clamped condition. Let ρ_{ji}^{-} denote the unconditional correlation between the states of neurons i and j (when the machine operates in the free-running condition). Then, according to the Boltzmann learning

rule, the change applied to the synaptic weight w_{ji} is defined by

$$\triangle w_{ji} = \theta(\rho_{ji}^{+} - \rho_{ji}^{-}), i \neq j. \tag{4.6}$$

where θ is a learning rate parameter.

4.2.2 Training Paradigms.

- **Supervised Learning.**

  An essential feature of supervised learning is the presence of an external teacher. A teacher possesses knowledge about an environment in the form of the input-output examples. When the teacher and an artificial system are both exposed to some input vector of the environment, the teacher is able to provide the learning system with desired (target) responses for that input vector. The learning system's parameters are adjusted under the influence of both the input vector and the error signal (defined in (4.1)). These adjustments are carried out in a step-by-step fashion, such that knowledge is gradually transferred from the teacher to the artificial system. This form of supervised learning is the error-correction learning. The values of the free parameters of the artificial system constitute a point on an error surface and at each learning step they are updated such that their values move in the direction towards a minimum of the error surface. This is done by means of the information regarding the gradient of the error surface. The negative gradient of the error surface at any of its points is a vector that points in the direction of the steepest descend. In fact, in the case of the supervised learning from examples the artificial system uses instantaneous estimates of the gradient vectors, where example indices substitute those of the time. In this case an operating point (a point on the error surface induced by the current values of free parameters) performs a random walk in the space of free parameters' values.

Supervised learning can be performed in an *on-line* or *off-line* manner. In the off-line case, a separate computational facility is used to design a learning system. Once a desired performance is achieved, the design is "frozen" and after this the artificial system operates in a static manner. In the on-line case, a learning procedure (algorithm) is implemented as a part of the artificial system itself and learning is conducted in *real time* which results in a dynamic behavior of the artificial system.

The disadvantage of supervised learning (in both on-line and off-line cases) is that it cannot learn new strategies which are not covered in the set of training examples.

- **Reinforcement Learning.**

  Reinforcement learning is conducted on-line with an objective to learn an input-output mapping through the process of trial and error and is designed such that a scalar performance index called a reinforcement signal is maximized.

  Consider a learning system interacting with an environment which is described by a dynamic process with a finite set of states X. At time steps $n = 0, 1, 2, ...$ the process is in states $\mathbf{x}(n) \in X$. The learning system can undertake some actions from the set A. It performs an action $\mathbf{a}(\mathrm{n}) \in \mathrm{A}$ after observing the state of the process $\mathbf{x}(\mathrm{n})$, which causes the process to transit to some state $\mathbf{x}(n+1)$. After this, the learning system receives a reinforcement signal $r(n+1)$, which serves as an indication of the correctness of the action performed by the learning system. It can be positive (reward), negative (punishment) or zero. The objective of the learning system is to find a policy for selecting a sequence of actions that is optimal in some statistical sense. Assuming that the process is initially in state $\mathbf{x}(0) = \mathbf{x}$, a measure of the learning system's performance is defined by the evaluation function

$$J(\mathbf{x}) = \mathrm{E}[\sum_{\mathbf{k=0}}^{\infty} \gamma^{\mathbf{k}} \mathrm{r}(\mathrm{k}+1) | \mathbf{x}(0) = \mathbf{x}] \tag{4.7}$$

where the expectation operator E is taken with respect to the policy used to select actions by the learning system. The summation term is called the cumulative discounted reinforcement. The factor $0 \leq \gamma < 1$ is called the discount-rate parameter. By adjusting this parameter we can control the extent to which the learning system is concerned with the long-term versus short-term consequences of its actions. The basic idea behind reinforcement learning is to learn the evaluation function $J(\mathbf{x})$, so as to predict the cumulative discounted reinforcement received in the future.

In reinforcement learning, the information contained in a reinforcement signal evaluates behavior of the learning system but does not indicate if the improvement is possible and how the system should change its behavior to improve performance. To obtain this information, the learning system probes the environment through the use of trial and error and delayed reward.

- **Unsupervised Learning.**

  In unsupervised or self-organized learning there is no external teacher to observe and guide the learning process. In other words, there are no specific examples to be learned by an artificial system. The system does not receive any external feedback concerning its performance, which is specific to the task that the system learns or the training data that it uses. A task-independent measure of the quality of representation, that the system is required to learn is used and free parameters of the artificial system are optimized with respect to this measure. Once the learning system has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input, hopefully in more explicit or simple form. It is hoped that the transformed version of the sensory input would be easier to interpret so that correct responses could be associated with the system's representation of the environment more quickly. The representatives of this paradigm are Hebbian and competitive learning discussed previously.

4.3 Identification of Cellular Automata. Review of the Existing Approaches.

As discussed in Section 2.5, cellular automata are not only a subject of theoretical research, they have already proven to be useful in many applied areas. However, as indicated in [44], in spite of the interesting and powerful features of cellular automata, their wide applicability as computational models has been limited because of the difficulty (and lack of the research) to design (program) cellular automata to perform specific tasks. The problem of identification (design) of cellular automata, capable of useful computation, has already been addressed by some researchers. In the rest of this section we will mention several interesting studies concerning this problem. We do not intend to give a detailed description of the methods already developed in this area, but rather outline applications where learning cellular automata were used and learning strategies employed to train them.

In a classical sense, cellular automata are considered to be autonomous systems. This means that they do not have any external inputs, i.e. their evolution (computation) is based on state transitions starting from some initial configuration of internal states on CA lattice. Any external influence on such systems is impossible, it is only possible to observe an evolution of cellular automata configurations in a sequence of discrete time steps. Similarly, there are many natural systems which are autonomous and can be modeled with cellular automata. Amongst them, there are such systems for which the local rules of interactions between cells of modeling cellular automata are not known and only snapshots of the CA configurations can be obtained from observations of the natural systems. In this case, identification of cellular automata transition rules can be very useful and in fact can be done providing that there are a sufficient number of snapshots of CA configurations.

The methodology to solve the problem of cellular automata identification from the fixed snapshots of successive CA configurations have been introduced in [1]. In

this approach, a set of cell states is assumed to be known but a size of the neighborhood and a transition rule of a cellular automaton must be identified. In addition, the goal is to obtain a minimal description of a cellular automaton (with respect to the size of the neighborhood) which simulates a system of interest. The author in [1] presents the algorithms for identification of cellular automata belonging to all classes introduced in Section 2.3 and analyzes complexities of these algorithms. The idea, that constitutes the basis of the identification algorithm for a deterministic cellular automaton, consists in constructing a table, representing a transition rule for the cellular automaton. This table consists of the strings of the form $< \alpha_{i_1}, ..., \alpha_{i_k} | \beta_i >, i \in \{1, ..., 2^k\}$, where k is a size of the neighborhood. The tuples $< \alpha_{i_1}, ..., \alpha_{i_k} >, i \in \{1, ..., 2^k\}$ correspond to all possible neighborhood configurations and each $\beta_i, i \in \{1, ..., 2^k\}$ represents the next state to which a cell should transit when its neighborhood state configuration is $\alpha_{i_1}, ..., \alpha_{i_k}$. Such a table is constructed while scanning through provided snapshots of successive CA configurations. At the beginning, the size of the neighborhood is assumed to be minimal ($k = 3$). The neighborhood state configuration of each cell is read from the CA configuration $C(t), t \in \{1, ..., p\}$. It is concatenated with the cell's next state, obtained from the CA configuration $C(t+1)$ and the resulting string $< \alpha_{i_1}, ..., \alpha_{i_k} | \tilde{\beta}_i >$ is added to the table. If the string, corresponding to the same neighborhood configuration $< \alpha_{i_1}, ..., \alpha_{i_k} | \hat{\beta}_i >$ is already present in the table, the states $\tilde{\beta}_i$ and $\hat{\beta}_i$ are compared. If they are different, it is concluded that the size of the neighborhood is too small to adequately describe the computation in the given CA configurations. It is increased and the table construction starts from the beginning. If the states $\tilde{\beta}_i$ and $\hat{\beta}_i$ are equal, nothing is added to the table and scanning of the CA configurations proceeds. In the case of the probabilistic and fuzzy cellular automata, the frequencies of transitions of cells with neighborhood configuration state $\alpha_{i_1}, ..., \alpha_{i_k}, i \in \{1, ..., 2^k\}$ to state $\beta_j, j \in \{1, ..., |Q|\}$ are calculated and based on this, transition probabilities and fuzzy grades are derived.

In [2], a hierarchy of synchronous stationary cellular automata is established:

totalistic CA $\subset$ deterministic CA $\subset$ probabilistic CA $\subset$ fuzzy CA. Algorithms (based on the same idea as previously discussed) are presented for two types of fuzzy cellular automata. If an identified cellular automaton appears to belong to the class, lower in the hierarchy than fuzzy CA, an identified fuzzy transition rule can be transformed (simplified) after the identification phase.

In [1], the applications of this identification technique are discussed in the contexts of discrete motion, cryptography, distributed intelligence, reaction-diffusion media, interacting populations and design of molecular computers. The discussion about these potential application areas in [1] has a rather illustrative manner and we are not aware of any instances where this technique has been implemented to solve practical problems. Nevertheless, the author presents a thorough analysis of the complexity of the proposed algorithms which indicates that they can be used as a feasible tool to solve certain cellular automata identification problems.

There may be an interpretation of computation in the context of cellular automata which is alternative to that which views CA as autonomous systems. In this case [44], a transition rule is interpreted as a "program", capable of performing a particular task. An initial configuration of a cellular automaton is considered to be an "input", and the cellular automaton runs for some specified number of time steps or until it reaches some goal pattern (possibly a fixed point). In this case, intermediate CA configurations (between initial and goal) may be not known and identification techniques, different from those discussed previously, should be used.

In [44], [43], a uniform one-dimensional binary circular cellular automaton was designed to perform a density classification task: decide whether or not an initial configuration consists of more that half ones. If so, the desired behavior of the cellular automaton was to relax after a certain number of time steps to a fixed point pattern of all ones, otherwise - all zeros. For identification of cellular automata capable of performing this task a genetic algorithm with mutation and crossover was employed. The population of transition rules (thus CAs) was generated and evolved with application of genetic operators. The rule for a cellular automaton with the

size of the neighborhood equal to 7 was found which could correctly classify about 95% of the initial configurations.

In [63] the same task was required to be performed by a cellular automaton, but in this case, by a non-uniform one. A genetic algorithm with mutation and crossover was also used as a tool to identify a cellular automaton. If the population of non-uniform CA had to be evolved, the search space would be much larger than in the case of the uniform cellular automata. This problem was approached in [63] by application of a local evolutionary process: the evolution took place at each cell locally. Fitness of a transition rule was evaluated for each cell, not globally for the entire CA. In this case in order to obtain a better rule for a particular cell, genetic operators were applied only to the fitter rules of the cells which were neighboring with that particular cell. Successful results were also obtained in this study. It was demonstrated that a non-uniform cellular automaton can attain high performance (94% correctly classified initial configurations) with the minimal size of the neighborhood ($k = 3$), in which case it outperformed the best uniform cellular automaton with the same size of the neighborhood. This result provides support to the hypothesis that non-uniform cellular automata reduce connectivity requirements as compared to uniform ones.

In recent years, one-dimensional 3-neighbor linear non-uniform cellular automata[1] (LNCA) have been proposed as an alternative to linear feedback shift registers in such applications as test pattern generation, pseudorandom number generation, cryptography, error correcting codes, signature analysis etc. LNCA are a special form of linear finite state machines (LFSMs). Every LFSM is uniquely represented by a transition matrix, which in turn has a characteristic polynomial. Characteristic polynomials represent certain properties of LFSMs. Finding a particular type of LFSM (e.g. particular LNCA) with a specific polynomial is a problem of interest in VLSI. In [15] an algorithm for identification of cellular automata, that have given characteristic polynomials, is presented. This algorithm is based on a correspon-

[1] A transition function of a linear cellular automaton involves only addition modulo 2 operation.

dence between CA characteristic polynomial computations and Euclid's greatest common divisor algorithm.

Based on the interpretation of cellular automata computations as input-output mappings, cellular automata were applied to the problems of image recognition [55]. Probabilistic cellular automata (PCA) are used for this task. Each cell of a probabilistic cellular automaton corresponds to a pixel on an image. Some number of "feature detectors" are associated with each cell, producing outputs in the range $[0, 1]$ which represents the confidence of each feature detector in the presence of its feature in the corresponding point of the image. The feature detectors' outputs define the configurations of features at each pixel, which are state variables of PCA dynamics. This dynamics is defined on PCA configurations by a Markov process, which provides the probabilities of transitions from one PCA configuration to another. The automaton is allowed to evolve for some number of time steps before it comes to its final configuration, representing the detected features of the image. A function that defines the probabilities of state transitions for the probabilistic cellular automaton is defined as a function of some parameters $J_d^{\alpha\gamma}$ where α, γ are some features and $d \in \{1, ..., k\}$ is an index of a cell in a neighborhood. These parameters can be viewed as the weights of the connections between a given cell and its neighbors. A positive large value of a parameter $J_d^{\alpha\gamma}$ indicates that having a feature γ at a cell d in the neighborhood of a cell m strongly supports the hypothesis of having a feature α at the cell m. These parameters are identified in the process of learning. The cost function that is being minimized in the learning process consists of the sum of correlation between the obtained and desired states of the cells in the final configuration. The results of the learning process were successful and learning appeared to be faster compared to the Boltzmann machine method.

In a [53] probabilistic cellular automaton was used to solve a combinatorial problem : partition a weighted graph into s subgraphs such that the weight of edges between subgraphs are minimized and subgraphs are balanced. In the context of this application, each cell corresponds to a vertex of a graph and s cell states correspond

to subgraphs into which the given graph should be partitioned. The fact that a cell is in some state $i \in \{1, ..., s\}$ indicates that a corresponding vertex belongs to the subgraph i. The actions performed by the cellular automaton via state transitions, are those that move a vertex from one subgraph to another. This constitutes the computation of the cellular automaton which is performed in a probabilistic manner. The probabilities of state transitions are determined in the process of learning via reinforcement technique. The computation of such a cellular automaton also depends on the neighborhood template which is determined in the learning process using a genetic algorithm with mutation and crossover operators.

A probabilistic non-uniform cellular automaton has been applied to the problem of controlling unstable systems [41], [54], [11]. A special probabilistic scheme was introduced to encode the states of a system, being controlled, into the CA initial configurations. An encoded system's state constitutes an input to a CA based controller, which computes a control action to be applied to the system depending on its state. A method to decode the a control action from a CA final configuration had been also designed. Since both system states and control actions are usually real valued variables, the mapping of their values into the discrete CA lattice is not apparent and special encoding/decoding schemes have to be used. Learning in a probabilistic cellular automaton based controller is accomplished in [54] by modifying the probability distribution function of cell state transitions to appropriately reflect the environmental performance feedback - reinforcement signal. The cost function that is minimized in the process of learning reflects a probability that the learning cellular automaton will receive penalty - negative reinforcement from the environment. A stochastic gradient descent method for minimization of the cost function is discussed in this study. This learning algorithm is shown to be convergent with specific conditions imposed on learning rate parameter. However, the simpler implementation of stochastic reinforcement learning algorithm is proposed in [41] and tested on the problems of controlling pendulum [41], [11], double pendulum [41] and continuous stirred tank reactor problem [11]. For all three applications, successful solutions

have been found and a cellular automaton based controller with a relatively simple structure was able to perform a control task in simulation experiments.

The area of learning systems based on cellular automata is still in its infancy as compared to, for example, artificial neural networks. Nevertheless, studies already conducted in this field demonstrate that cellular automata can be "trained" to perform specified tasks using various techniques from machine learning. The efficiency of learning cellular automata devices in various applications will have to be studied more extensively and compared with other artificial intelligence approaches.

Chapter 5

Problem of Estimating an Unknown Function.

5.1 Problem Statement.

A problem of predictive/estimating learning is very simple to formulate. Knowledge about a system is given by several (possibly many) measurable quantities, called variables. The variables are divided into two groups: input and output variables. The goal of the learning is to develop a computational relationship between the inputs and the outputs (formula/ algorithm) for estimating the values of output variables given only the values of input variables.

Consider, as an example of a system of interest, a manufacturing process. The input variables would be the various parameters that control the process (e.g. chemical concentrations, processing time, etc.). The output variable would be the quality of the final product. The goal in this case would be to estimate the quality, given the values of the input parameters, without actually running the manufacturing process. Realizing this goal could bring considerable financial benefits.

Thus, a learning system is a computer program that constructs a rule for estimating the values of output variables of a real system, given the values of input variables of this system. In contrast to expert systems, which attempt to organize the knowledge of human experts in a particular field, predictive/estimating learning systems attempt to build useful estimating rules purely by processing data obtained in the past: cases for which the values of both input and output variables have been determined. A generic learning system does not contain any domain specific knowledge; all useful information is assumed to be contained in past data.

Both the input and output variables can be of two fundamental types: real and categorical. A real valued variable assumes values over some subset of real line R^1. These values have an order relation and distance defined between any two values. For categorical variables, the values are certain labels (e.g. brand names, type of disease, nationality etc.). If a categorical variable assumes only two values, they then can be mapped to the real values of 0 and 1 and this variable can be further treated as a real one without loss of generality. When a categorical variable assumes more than two (say P) values, then it can be converted into P real valued variables (which assume the values of 0 and 1) - one real variable for each categorical value. This technique is referred to as "dummy variables" in statistics. Categorical outputs occur often and represent an important class of problems referred to as pattern recognition in engineering and classification/discriminant analysis in statistics. The dummy variable technique can always be used in these cases and from a theoretical point of view, it is sufficient to consider only real valued outputs.

A general mathematical model of an input/output mapping under the study is

$$y_k = g_k(x_1, ..., x_n, z_1, ..., z_m), k = 1, ..., K \tag{5.1}$$

Here y_k is the k^{th} output variable and g_k is a (real) single valued deterministic function of all possible (observed and unobserved) inputs that contribute to the change of the values of y_k, where $x_1, ..., x_n$ refer to observed inputs and $z_1, ..., z_m$ - unobserved ones. Uncertainty, present because of the unobserved inputs, can be

represented by a statistical model

$$y_k = f_k(x_1, ..., x_n) + \epsilon_k, k = 1, ..., K \tag{5.2}$$

where f_k is a function of the observed inputs only and an additional random component ϵ_k is added to reflect the fact that the observed inputs do not uniquely specify the output value.

The fundamental models (5.1) and (5.2) can be formed for each of the outputs of a system and treated as separate problems:

$$y = f(x_1, ..., x_n) + \epsilon \tag{5.3}$$

This is the most common way to proceed in the cases of multi-output systems, however some strategies that exploit association between outputs were also proposed [18].

The goal of any learning approach is to obtain a useful approximation $\tilde{f}(x_1, ..., x_n)$ of the target function $f(x_1, ..., x_n)$ (5.3). Supervised learning methods attempt to learn an input/output relationship by examples through a teacher. Teacher observes a system under study measuring the values of both (observed) input and corresponding output variables. Doing this repeatedly, the teacher collects L training samples - input-output values.

$$\{y_i, x_{i1}, ..., x_{in}\} = \{y_i, \mathbf{x_i}\}, i \in \{1, ..., L\} \tag{5.4}$$

The observed input values of the system, $\mathbf{x_i}$, are also the inputs to the artificial learning system that produces an output $\tilde{f}(\mathbf{x_i})$ in response to these inputs. Then the teacher provides an error between the artificial and real systems' outputs $\tilde{f}(\mathbf{x}_i) - y_i$ and the artificial system modifies the input/output relationship $\tilde{f}(\mathbf{x})$ in accordance with this error. Thus, supervised learning is employed in function estimation. The hope is that in the process of learning, the artificial system will be able to generalize the information contained in the training samples and after the completion of the learning process, it will be able to produce the satisfactory estimates of the target function for the inputs not contained in the training set.

5.2 Search for a Solution.

As one can see, the problem of supervised learning is very simple to state, but thus far an absolutely general useful solution to it has not been found. The main source of difficulty is a finite size of a training set in all practical applications. In the case of a training set of a finite size L, the supervised learning problem consists in finding a solution to

$$\tilde{f}(\mathbf{x}) = \min_{\mathbf{g}(\mathbf{x})} \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{L}} [\mathbf{y_i} - \mathbf{g}(\mathbf{x_i})]^2 \tag{5.5}$$

and the solution is not unique. There are in fact an infinite number of functions that can interpolate L data points yielding the minimum value for the criterion function (5.5). If the noise is absent ($\epsilon = 0$ in (5.3)) then a target function $f(\mathbf{x})$ will be amongst the solutions. If the noise is present, none of the solutions of (5.5) will be exactly $f(\mathbf{x})$. Thus (5.5) represents an ill-posed problem; in both noisy and noiseless cases, there is no unique solution.

In order to obtain some useful results for finite size L, one must restrict the eligible solutions of (5.5) to some set, smaller than all possible functions. This restriction is imposed by a user based on considerations outside the training data. This is usually done implicitly by the choice of a learning method. However, this approach does not make the problem less ambiguous. Imposing a particular restriction may lead to finding a unique solution for (5.5), but then there is an infinite number of restrictions, each of which yields some solution; the multiplicity of solutions has been only transferred to a different mechanism.

Many of the well known learning methods restrict their solution spaces to a particular class of functions characterized by a set of parameters $\{w_1, ..., w_N\}$

$$\tilde{f}(\mathbf{x}) \in \{h(\mathbf{x}|w_1, ..., w_N)\} \tag{5.6}$$

and

$$\tilde{f}(\mathbf{x}) = h(\mathbf{x}|\hat{w_1}, ..., \hat{w_N}) \tag{5.7}$$

for some particular setting of the parameter values $\langle \hat{w}_1, ..., \hat{w}_N \rangle$. The problem in (5.5) becomes:

$$\langle \hat{w}_1, ..., \hat{w}_N \rangle = argmin_{\langle \hat{w}_1, ..., \hat{w}_N \rangle} \sum_{i=1}^{L} [y_i - h(\mathbf{x}|w_1, ..., w_N)]^2 \tag{5.8}$$

with $\tilde{f}(\mathbf{x}) = h(\mathbf{x}|\hat{\mathrm{w}}_1, ..., \hat{\mathrm{w}}_\mathrm{N})$. If $\{h(\mathbf{x}|w_1, ..., w_N)\}$ is a class of continuous functions and parameters themselves take on continuous real values, then the solution to (5.8) can be obtained by numerical optimization techniques.

One method to further restrict the possible solutions to (5.8), very popular in the neural networks community, is to define a particular path ("curve") through the N-dimensional parameter space, and restricting solutions to those that lie on the specified path. One way to specify this path is through a set of differential equations $\frac{\delta I}{\delta w_j}, j = 1, ..., N$, where $I(w_1, ..., w_N)$ is another specified function of the parameters - a cost function. In addition, it is necessary to specify a starting point for the parameters - $\langle w_1^{(0)}, ..., w_N^{(0)} \rangle$. Then, for example, the gradient of the function $I(w_1, ..., w_N)$ determines the particular path. Beginning at the starting point of the parameters, a small step is taken in the direction of the negative gradient of the cost function. The gradient is evaluated in a new point so obtained and a new step is taken along a new gradient direction. These steps are continued until the gradient becomes zero, indicating a local minimum of $I(w_1, ..., w_N)$ or until some other stopping criteria are satisfied.

The described method is a first order optimization technique applied to a cost function with respect to the free parameters of the learning system. There are many possibilities for selection of a cost function, the mean-squar-error being the most popular one:

$$\varepsilon_{av} = \frac{1}{2L} \sum_{\ell=1}^{L} e^2(\ell) \tag{5.9}$$

where L is the number of training examples used in the learning process. In this case the gradient is calculated after all training examples were presented to the learning system and only then do the adjustments of the free parameters take place. This

training mode is known as a *batch mode*. Alternatively, one may use a *pattern-by-pattern mode* of training, in which case free parameters of the artificial system are updated after the presentation of each training example. The cost function used in this case is

$$\varepsilon(\ell) = \frac{1}{2}e^2(\ell) \tag{5.10}$$

at presentation of the ℓ^{th} training example. A learning process designed to follow a certain path consists of many presentations of the training set. One presentation of the entire training set is called an *epoch*. In the pattern by pattern training mode, the order of presentation of training examples is often randomized from one epoch to the next. This randomization tends to make a search in the parameter space stochastic over the training epochs. One of the reasons that favors this approach is that it may help to escape a local minimum of the cost function. But at the same time a good local minimum can be missed because of the mutual interference of the parameters' changes (caused by different patterns). Subsequent gradient directions may interfere: minimization in a direction a followed by minimization in a direction b does not imply that the function is minimized on the subspace generated by a and b. Minimization along the direction b may in general "spoil" minimization along the direction a. This is why the minimization iterations (epochs) have to be performed a number of times much larger than the number of free parameters.

The concept of non-interfering directions is the basis of another method for optimization - conjugate gradient method [6]. Assume that last step of minimization has been performed in the direction p_i and a new point in the parameter space has been obtained. At this point a new minimizing direction p_{i+1} is chosen such that it is perpendicular to the previous one - p_i, and the previous minimization is not spoiled. For a quadratic function, the conjugate gradient method is guaranteed to converge in at most $(N + 1)$ function and gradient evaluations. For a general function it is necessary to iterate the method until a suitable approximation to the minimum is obtained. In practice, the number of steps is much smaller than that required for minimization by steepest descent method.

The conjugate gradient method is a second-order optimization technique (uses second-order derivatives). Other second-order methods include those based on Newton's method. Amongst these methods are: method of line searches, model-trust-region methods, secant methods, Gauss-Newton method, Levenberg-Marquardt method etc. The difficulty in applying of these methods lies in the necessity to calculate the Hessian matrix (matrix of second-order partial derivatives) which can be very large and extremely difficult to calculate analytically. In this case the analytic Hessian matrix has to be approximated, but it is important that the approximations used maintain the symmetry and positive-definiteness properties of the Hessian. This can be achieved, for example, by application of the Choletsky factorization technique [27]. Most of the second order methods require a large amount of computation per iteration (of order $O(N^2)$ or $O(N^3)$) and thus are more suitable for problems with limited number of free parameters ($N < 100$). However these methods not only speed-up the convergence of the optimization procedure but also make the learning algorithm more robust with respect to some parameters of the algorithm (e.g. learning rate parameter). In addition, application of these methods usually leads to the solutions with higher degree of precision.

5.3 Objectives of Function Identification Through Supervised Learning.

There are two distinct objectives that can be attained by the application of supervised learning: estimation and interpretation. In the case of estimation, it is expected that in the future new observations of the input variables will be encountered but no corresponding output values will be available and a mechanism for their estimation would be extremely useful. In this case, a primary goal of a learning system is the accuracy of the estimates.

Another reason for applying supervised learning is interpretation. In this case, an

approximating function $\tilde{f}(\mathbf{x})$ is produced with the goal to get an understanding and insight into the mechanism that produced the data; no future data for estimation is necessarily envisioned. For example, one may be interested to know which of the input variables are the most relevant, what is the dependence of the output on the most relevant input variables etc. Such information can be very useful in understanding how the system works and perhaps how it can be improved. In this case, accuracy is not the only goal for the learning system. The ability of the artificial system to present the knowledge, gained in the process of learning, in the way in which a person can understand and interpret this knowledge is of main importance. The major successful methods for supervised learning differ greatly in the extent to which they can achieve this goal and only a few provide highly interpretable results.

The problem of a computer learning has been initially studied in the fields of applied mathematics (function approximation), statistics (regression and classification) and engineering (pattern recognition). Later this problem became one of the research subjects of artificial intelligence (machine learning) and biologically motivated methods for data modeling - connectionist learning systems (CLS). Statistics is one of the oldest disciplines to study data based learning and has developed numerous approaches and theories to solve this learning problem. CLS, on the other hand, is one of the most recent popular techniques, whose theoretical background is not yet developed as much as that of statistics. The objectives and methodologies employed by these two fields are rather different. It is important to be aware of these differences when making an attempt to compare the efficiencies of various methods. Some basic differences of statistical and connectionist learning approaches to function estimation were summarized in [18]:

- *Model complexity.* CLS usually have higher complexity (number of free parameters) than statistical methods.
- *Goals of modeling.* In statistics the usual goal is interpretability, which favors structured models (i.e. classification trees and liner regression). In CLS re-

search the primary goal is generalization/estimation, and CLS usually have little, if any, interpretability. However, for many high-dimensional problems even structured models are very difficult to interpret.

- *Batch versus pattern-by-pattern processing.* Most statistical methods utilize the whole training set before making any adjustments to the model they produce (batch mode), whereas CLS methods favor iterative processing of training samples (pattern-by-pattern mode). These methods require multiple presentation of the same training data to a learning system and is usually slower than statistical methods.

- *Computational complexity and usability.* Since statistical methods extract information from the entire training set, they tend to be more computationally complex and difficult to use by non-experts in statistics. CLS methods on the other hand are computationally simpler and understood more easily by novice users.

- *Robustness and quality of estimates.* CLS appear to be more robust than statistical ones with respect to the tuning of the learning algorithm's or model's parameters. Another important aspect is the quality of estimates providing their confidence intervals. Confidence intervals are routinely provided in statistical methods but not in CLS. Because of the "black box" structure of CLS, the quality of the solution can not be guaranteed.

- *Search for the best method.* Most statistical and CLS methods are asymptotically good, i.e. they can guarantee good estimates when the number of training samples grows very large. Unfortunately, real problems provide only finite and usually sparse data sets for learning. In this case, asymptotic performance is irrelevant and no single method dominates all others for all possible data sets. The real research goal is not to find the best method, but to characterize the classes of functions (together with some assumptions about the noise, smoothness etc.) for which a given method works best.

5.4 Selected Methods of Function Estimation.

Most of the learning methods can be viewed as *dictionary* methods. A *dictionary* method uses a set of predefined functions - a dictionary $\{b(\mathbf{x}|\gamma)\}_\gamma$, where $\gamma = \{\gamma_1, ..., \gamma_N\}$ is a set of parameters that characterize and uniquely index functions in the dictionary. The goal of function estimation/approximation in this case is to find a subset of functions from the dictionary $\{b(\mathbf{x}|\gamma_m)\}_{m=1}^M$ whose linear combination

$$\tilde{f}(\mathbf{x}) = \sum_{m=1}^{M} a_m b(\mathbf{x}|\gamma_m) \tag{5.11}$$

most accurately approximates the target function $f(\mathbf{x})$. The number of functions M is considered to be either fixed or serves as a model selection parameter.

A dictionary is said to be complete with respect to a class of the target functions if any target function in this class can be exactly represented by the entire dictionary:

$$f(\mathbf{x}) = \sum_{m=1}^{\infty} a_m^* b(\mathbf{x}|\gamma_m) \tag{5.12}$$

for some set of coefficients $\{a_m^*\}_1^\infty$. The corresponding method is said to be a universal approximator for that class of target functions.

The methods described in the following are universal approximators for the class of continuous target functions. The methods discussed below differ primarily in their choice of the dictionary. Thus, each of the methods is especially effective for the functions of the class that can be efficiently represented by the functions in the dictionary. By efficient representation we mean that it involves a relatively small number of functions from the dictionary. No method thus is superior to all others over all target functions.

The methods discussed below represent only a small part of those that have been developed so far, however they are amongst the most popular ones. Many of the other existing methods can be regarded as variations on the methods discussed here. A thorough comparative study of the methods belonging to the groups discussed above can be found in [17].

1. **Projection Pursuit and Feed-Forward Neural Networks.**

The dictionaries of these methods consist of the functions of the form

$$\{b(\mathbf{x}|\gamma,\gamma_0) = s(\gamma_0 + \gamma^t x)\}_{\gamma,\gamma_0} \tag{5.13}$$

where $\gamma^t\mathbf{x} = \sum_{j=1}^{n} \gamma_j x_j$ is the linear combination of the input variables. In the case of neural networks, the most popular choice of a single-variable function $s()$ is a *sigmoid* function (e.g. logistic, arctangent etc.). The simplest model of the neural network is a three-layer feed-forward neural network. The first layer consists of the elements (neurons) that correspond to the input variables; the second (*hidden*) layer consists of the neurons that compute the dictionary functions, and finally the output layer computes the linear combination of the outputs of the neurons in the hidden layer and represents the output. Additional hidden layers can be added to the network as well, whose neurons compute the functions of the same dictionary as the first hidden layer. Adding an additional layer to the network enlarges the dictionary by involving more parameters. The effect of this on the performance of the network depends on how well the resulting dictionary suits the target function.

In the case of projection pursuit, a function $s()$ is more flexible; it has the form $s_\alpha(\gamma_0 + \gamma^t x)$, where $\alpha = \langle \alpha_1, ..., \alpha_p \rangle$ are additional parameters that characterize a wide class of single argument functions. These parameters are identified in the process of learning (simultaneously with parameters (γ, γ_0)) to best fit the training data. This increases the number of functions in the dictionary and thus the projection pursuit method has less bias concerning the representation of a target function.

2. **Radial Basis Functions.**

The dictionary for this method is comprised of the functions of the form;

$$\{r[((\mathbf{x}-\mathbf{t})^t\mathbf{H}(\mathbf{x}-\mathbf{t}))^{\frac{1}{2}}]\}_{\mathbf{t},\mathbf{H}} \tag{5.14}$$

The parameters that characterize the functions of this dictionary are the n elements of the vector $\mathbf{t} \in \mathrm{R}^n$ and $\frac{n(n+1)}{2}$ parameters associated with nonnegative

definite matrix $H \in R^{n \times n}$. A popular choice for the single variable function $r(z)$ is the Gaussian function $r(z) = \exp(-z^2)$. This dictionary thus has $\frac{n(n+3)}{2}$ parameters and is rather large. Its size can be reduced by placing additional constraints on the matrix H. Often it is required that the matrix H be a multiple of the $n \times n$ identity matrix $H = \rho^2 I_n$. This reduces the number of parameters to $n + 1$.

3. **Recursive Partitioning Tree-Structured Methods.**
The methods of this family have dictionaries of the form

$$\{I(\mathbf{x} \in \mathrm{R})\}_{\mathrm{R}} \tag{5.15}$$

where $I(\nu)$ is an indicator function of the logical argument ν

$$I(\nu) = \begin{cases} 1 & \text{if } \nu \text{ is true} \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

The symbol R here represents some subregion $R \in R^n$ and is characterized by a set of parameters (e.g. a hyper-rectangle). In recursive partitioning the subregions are disjoint such that for any set of input values $\mathbf{x}$, only one indicator function is nonzero. In this case,

$$\mathbf{x} \in \mathrm{R_m} \Rightarrow \tilde{\mathrm{f}}(\mathbf{x}) = \mathrm{a_m} \tag{5.17}$$

where $\{R_m\}_1^M$ represent the entries for which the indication function is true. The approximation function is thus a piecewise constant function.

The corresponding optimization problem is combinatorial rather than numerical. If M is relatively large it is a formidable problem and it is solved with greedy optimization strategies based on recursive partitioning. We will briefly describe one method employed in [12] (CART method). The initial region R_0 consists of the entire input space. This region is optimally divided into two regions R_1 and R_2 by a split on the input variable x_v at a split point t. The values of $v \in \{1, ..., n\}$, where n is the number of variables, and t are chosen such that when the indicator function $I(\mathbf{x} \in \mathrm{R_0})$ is replaced by two functions

$I(\mathbf{x} \in \mathrm{R}_1)$ and $I(\mathbf{x} \in \mathrm{R}_2)$ in (5.11) the best fit to the data is achieved. Each of these newly created regions R_1 and R_2 is split again. This recursive procedure is repeated until a large number of the regions are created. These regions are then optimally combined with their adjacent regions to form a smaller set to be used in the approximation. Usually this recombinational process is based on some model selection method - a method intended to select the model with the least complexity for the required accuracy (e.g. cross-validation).

The piecewise nature of the resulting approximation function limits their accuracy on the continuous smooth target functions. However, one area where recursive partitioning methods tend to dominate all others is in interpretability. The recursive partitioning can be represented graphically as a binary tree. The leaves of this tree are associated with the final partition and are labeled by the approximating values a_m. The internal nodes are labeled by the splitting variables x_v and the split points t. Thus this binary tree contains all the information associated with the approximation in an easy interpretable graphic form.

4. **Tensor Product Methods.**

The functions included into the dictionaries of these methods are of the form

$$\{\prod_{j=1}^{n} \psi(x_j|\gamma_j)\}_{(\gamma_1,\ldots,\gamma_n)} \tag{5.18}$$

Each of these functions is a tensor product of functions of a single variable - one of the inputs, and is characterized by one or more parameters γ_j. Constant functions $\psi(x_j|\gamma_0) = 1$ associated with one or more input variables are allowed in the product. In this case the dictionary can be expressed in an equivalent form

$$\{\prod_{k=1}^{K} \psi(x_{v(k)}|\gamma_k)\}_{(K,\{v(k),\gamma_k\}_{k=1}^{K}} \tag{5.19}$$

where K is the number of non-constant factors and is one of the parameters characterizing a function in the dictionary.

The goal of the tensor product methods is to select the products (5.19) with a relatively small number of factors. The choice of the input variable set $\{v(k)\}_{k=1}^{K}$ and corresponding parameter values $\{\gamma_k\}_1^K$ incorporates the mechanism of recursive partitioning methods. However, tensor product methods employ continuous and more flexible functions $\psi(x_{v(k)}|\gamma_k)$ than those employed in recursive partitioning methods. This allows more accurate approximations of continuous functions. One choice for functions $\psi(x_{v(k)}|\gamma_k)$ can be spline functions. In addition individual factors can be functions from different parametric spaces. This enables tensor product methods to handle input variables of different types - real valued and categorical - more naturally. Tensor product methods are also known to deal well with the missing valued inputs.

5. **Fuzzy Approximating Systems.**

Fuzzy systems are widely used for the function identification problem in today's research. One of the approaches [72] involving fuzzy modeling can be also considered as a dictionary method. In this case, fuzzy systems have a form of series expansions of fuzzy basis functions, which are algebraic superpositions of fuzzy membership functions. The dictionary of this method consists of the functions of the form:

$$\{\frac{\prod_{i=1}^{n}\mu_{A_i^j}(x_i)}{\sum_{j=1}^{M}\prod_{i=1}^{n}\mu_{A_i^j}(x_i)}\}_{A_i^j, i\in\{1,...,n\}, j\in\{1,...,M\}} \tag{5.20}$$

where

$$\mu_{A_i^j}(x_i) = a_i^j \exp(-\frac{1}{2}(\frac{x_i - \bar{x_i^j}}{\sigma_i^j})^2); j \in \{1, ..., M\} \tag{5.21}$$

and $a_i^j, \bar{x_i^j}$ and σ_i^j are some real valued parameters. The functions $\mu_{A_i^j}$ are membership functions of fuzzy sets defined on the space of input variables. The most important advantage of using fuzzy basis functions is the possibility to include linguistic fuzzy rules, obtained from the human experts, into the training data. These linguistic rules have a great value [72] since they often contain information which is not present in the training input-output pairs. Using a fuzzy expansion, one can combine two sets of fuzzy basis functions

- one derived from input-output pairs and the other obtained from linguistic rules - into one fuzzy expansion.

Other important properties associated with approximation methods include training (computational) speed and the ease of use. Specific implementations of the methods almost always include some "tuning" of parameters to improve the performance of a method for a particular application. Some of these parameters can be really crucial for the efficiency of an algorithm. Thus, often there is no way to separate the power of a method from the expertise of a user. In this context the robust methods are those that are not very sensitive to the parameter settings. Tree-structured methods, adaptive splines and to some degree projection pursuit methods are more robust than the others.

Chapter 6

Fuzzy Cellular Automata.

6.1 Definition of a Fuzzy Cellular Automata Model.

A fuzzy cellular automaton (FCA) can be defined by a tuple $\langle \mathcal{L}, Q, u, f \rangle$, where

- $\mathcal{L}$ is a d-dimensional finite lattice on M cells, $d \geq 1$, $M \geq 2$.
- Q = {T, F} is a set of fuzzy states that each cell in $\mathcal{L}$ can take. Each cell x actually can take a value c in the unit interval [0,1]; then the degree to which a cell x is in states T and F can be calculated with the membership functions $\mu_T(x) = c$ and $\mu_F(x) = 1 - c$ respectively.
- u is a neighborhood template of cardinality k. The number of possible (fuzzy) neighborhood state configurations is thus 2^k. Let us denote a fuzzy neighborhood state configuration by $\alpha_i = (\alpha_{i1}, ..., \alpha_{ik}) \in \{T, F\}^k, i \in \{1, ..., 2^k\}$. Fix a cell x and let its neighborhood consist of cells $u(x) = (y_1, ..., y_k)$, where y_j currently has value c_j. Then the state configuration of the neighborhood $u(x)$ is in a fuzzy state α_i to the degree calculated with a membership function

$$\mu_{\alpha_i}(u(x)) = \min_{j \in \{1,...,k\}} \{\mu_{\alpha_{ij}}(y_j)\} \tag{6.1}$$

of the corresponding set α_i. This intuitively corresponds to the conjunction of the fuzzy conditions "cell y_j is currently in state α_{ij}", and formally to the aggregation operation on k fuzzy sets α_{ij} by means of the standard fuzzy intersection. Of course in the classical case, μ_{α_i} would take the value 1 for exactly one α_i and value 0 everywhere else.

- f is a (fuzzy) transition function, $f : [0,1]^k \rightarrow [0,1]$ determining the next state of a cell x given the state configuration of its neighborhood. According to this function, the next state of the cell x will be T to the degree calculated with the membership function

$$f(u(x)) = \max_{i \in \{1,\ldots,2^k\}} \{\min\{w_i, \mu_{\alpha_i}(u(x))\}\} \tag{6.2}$$

where the w_i's are free parameters in the interval $[0,1]$. This intuitively corresponds to taking a (weighted) disjunction of the fuzzy conditions "the neighborhood of cell x is currently in configuration α_i".
Function f is considered to be defined if its free parameters w_i are defined.

6.2 Discussion on the Form of the Transition Function.

According to the generic idea of a CA computation, a cell makes a transition to the next state on the basis of the current state of its neighborhood. For each of the possible neighborhood state configurations, there should be a designated state to which a cell should make a transition at the next time step. In the fuzzy model of CA, where cells can take values $c \in [0,1]$, a neighborhood configuration of any given cell matches all of the possible neighborhood state configurations $\{T, F\}^k$, but to different degrees. Intuitively, a transition function of a fuzzy cellular automaton should aggregate these degrees and thus it may be an aggregation operation (3.4) applied to the values $\mu_{\alpha_i}, i \in \{1,\ldots,2^k\}$. At the same time, the choice of the next

state of a cell should be mostly determined by the fuzzy neighborhood state configuration with the maximum degree μ_{α_i}. Clearly, in this case, such a transition function should be nondecreasing with respect to the values of μ_{α_i}. More than that, it should satisfy the following two properties for all $i \in \{1, ..., n = 2^k\}$:

$$h(\max(a_1, b_1), ..., \max(a_n, b_n)) = \max(h(a_1, ..., a_n), h(b_1, ..., b_n)) \tag{6.3}$$

$$h_i(h_i(a_i)) = h_i(a_i), \tag{6.4}$$

where $h_i(a_i) = h(0, ..., a_i, ..., 0)$, and a_i and b_i are some values of μ_{α_i}. Although in the context of our application it is practically impossible for all values $\mu_{\alpha_i}, i \in \{1, ..., n = 2^k\}$ to be equal to 0 or 1 at the same time, we may assume that at these extreme cases the function satisfies Axiom h1. Based on these properties and also requiring the transition function to be continuous, we can formally derive the form of the aggregation operation by means of which our transition function can be represented. The following theorem can be found in [37].

Theorem. Let $h : [0, 1]^n \rightarrow [0, 1]$ be a continuous function that satisfies Axiom h1, Axiom h2 and properties (6.3) and (6.4). Then,

$$h(a_1, ..., a_n) = \max(\min(w_1, a_1), ..., \min(w_n, a_n)) \tag{6.5}$$

where $w_i \in [0, 1]$ for all $i \in \{1, ..., n\}$.

Proof: First observe that $h(a_1, a_2, ..., a_n) = h(\max(a_1, 0), \max(0, a_2), ..., \max(0, a_n))$. Then by property (6.3) we can obtain $h(a_1, a_2, ..., a_n) = \max(h(a_1, 0, ..., 0), h(0, a_2, ..., a_n))$. By induction we can replace $h(0, a_2, ..., a_n)$ with $\max(h(0, a_2, 0, ..., 0), h(0, 0, a_3, ..., a_n))$. Repeating the same replacement we eventually obtain

$$\begin{aligned} &h(a_1, a_2, ..., a_n) = \max[h(a_1, 0, ..., 0), h(0, a_2, 0, ...0), ..., h(0, 0, ..., a_n)] = \\ &\max[h_1(a_1), h_2(a_2), ..., h_n(a_n)]. \end{aligned} \tag{6.6}$$

It remains to prove that $h_i(a_i) = \min(w_i, a_i)$ for all $i \in \{1, ..., n = 2^k\}$. Since the function h is continuous and nondecreasing, so is each $h_i(a_i)$. Also $h_i(0) = 0$ by Axiom h1. Let $h_i(1) = w_i$; then the range of h_i is $[0, w_i]$. For any $a_i \in [0, w_i]$,

there exists b_i such that $a_i = h_i(b_i)$ and hence, $h_i(a_i) = h_i(h_i(b_i)) = h_i(b_i) = a_i = \min(w_i, a_i)$ by (6.4). Also for any $a_i \in (w_i, 1], w_i = h_i(1) = h_i(h_i(1)) = h_i(w_i) \leq h_i(a_i) \leq h(1) = w_i$ and consequently, $h_i(a_i) = w_i = \min(w_i, a_i)$.

This theorem gives us exactly the form of the transition function defined in the previous section.

Chapter 7

Learning with Fuzzy Cellular Automata.

7.1 Problem Statement.

There is a real-valued concept (function) $C : [0,1] \rightarrow [0,1]$ that has to be identified (or estimated). The information about this concept is available in the form of a finite set of input - output pairs, $Tr(C)$, which is called a training set: $Tr(C) = \{(Input^{(1)}, Output^{(1)}), ..., (Input^{(L)}, Output^{(L)})\}$. The objective of identification through supervised learning is to create an artificial model of the concept C. During the learning process, the artificial learning system should generalize the information in $Tr(C)$, such that it will represent general knowledge about the concept. In other words, after learning is completed, the model, when presented with an input *Input*, which does not belong to $Tr(C)$, will be able to produce an output *Output*, that will match the concept's response to *Input* with some accuracy ϵ.

A tool, that we will use to acquire and represent the knowledge about the concept, is the 1-dimensional fuzzy cellular automaton (FCA) defined in the previous chapter. We will consider the neighborhood template $u(x)$ to be known *a priori*. A local

transition function, that will make the FCA model a given concept, is not known and has to be identified in the process of learning. Each input value of the concept is encoded into the FCA initial configuration by some method (a possible methods of encoding will be discussed in Chapter 8). Having been presented an input value, i.e. set to an initial configuration encoding an input value, the FCA then performs $T \geq 1$ global transitions. T is set to some constant value at the beginning of the computation. It is one of the tunable parameters of the system which can influence the ability of the FCA to represent certain continuous functions. After T global transitions, an output value is decoded from the FCA output configuration as follows:

$$y = \frac{1}{M} \sum_{m=1}^{M} \varphi(x_m(T)) \tag{7.1}$$

where M is a number of cells in the FCA and $x_m(T) \in [0, 1]$ is a value assumed by a cell m after the transition T, and

$$\varphi(x) = \frac{1}{1 + \exp(-\sigma(x - 0.5))} \tag{7.2}$$

The objective of the learning process is to identify a local transition function f, such that the fuzzy cellular automaton with this transition rule will model a given concept. Recall that a fuzzy transition function (6.2) is defined if the parameters $w_i, i = 1, ..., 2^k$ are defined. Therefore, our learning algorithm will search the parameter space for suitable values of these parameters. No *a priori* knowledge about these values is assumed to be available.

7.2 Sketch of the Learning Algorithm.

A learning process is conducted in an iterative manner. At the beginning of learning, values of the free parameters $w_i, i = 1, ..., 2^k$ are randomly initialized in the interval [0,1]. We are provided with a set of training examples $Tr(C)$ of size L, which represent input-output pairs of the target concept C. At each iteration $\ell, (\ell =$

$1, ..., L$), an input - output pair $(Input^{(\ell)}, Output^{(\ell)})$ is selected from the training set $Tr(C)$ and the input is presented to the FCA. The input value $Input^{(\ell)}$ is encoded into an initial FCA configuration and the FCA makes $T \geq 1$ global transitions. An output value is decoded using (7.1) and (7.2). This value is compared with the actual concept's output $Output^{(\ell)}$ and an error is obtained as $\varepsilon^{(\ell)} = y^{(\ell)} - Output^{(\ell)}$. This error is used to update the values of parameters $w_i, i = 1, ..., 2^k$ after each presentation of a training example - a pattern-by-pattern training mode is utilized. The learning process consists of multiple epochs (presentations of entire training set to the FCA) and proceeds until the average squared error over the entire training set converges to some minimum value (a stopping criterion is introduced later). During each epoch, training examples are drawn from the same training set $Tr(C)$ but presented to an FCA in a different (random) order.

7.3 Learning Algorithm.

Let us consider one iteration of the learning algorithm, say ℓ. An input value $Input^{(\ell)}$ was presented to the FCA and it produced an output value $y(\ell)$. The history of the FCA transitions is recorded and presented as follows:

$$\begin{array}{cccc} x_1^{(\ell)}(0) & x_2^{(\ell)}(0) & \cdots & x_M^{(\ell)}(0) \\ x_1^{(\ell)}(1) & x_2^{(\ell)}(1) & \cdots & x_M^{(\ell)}(1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(\ell)}(T) & x_2^{(\ell)}(T) & \cdots & x_M^{(\ell)}(T) \end{array}$$

The FCA evolved to the output configuration in T time steps - in T transitions. As previously mentioned, we will compute an error between an FCA's output and an actual concept's output as

$$\varepsilon^{(\ell)} = y^{(\ell)} - Output^{(\ell)} \tag{7.3}$$

Let us define a cost function

$$I = E[\frac{1}{2}(\varepsilon^{(\ell)})^2] \tag{7.4}$$

where E is the statistical expectation operator. We do not have any information on the probability distribution of $\varepsilon(\ell)$, thus we can only use an instantaneous estimate of the cost function:

$$I(\mathbf{w}^{(\ell)}) = \frac{1}{2}[\varepsilon^{(\ell)}]^2 \tag{7.5}$$

where $\mathbf{w}^{(\ell)}$ is a vector of values of the free parameters at the beginning of the iteration ℓ. The objective of the learning process is to minimize the cost function (7.5) with respect to the parameters $w_i, i = 1, ..., 2^k$. We will use the *steepest descent* optimization technique to conduct minimization. The negative gradient of the cost function points in the direction of its steepest descent. At each iteration ℓ of the algorithm (presentation of a training example), we will calculate a gradient vector of the cost function (with respect to free parameters) at the current operating point as defined by the current values of free parameters $w_i^{(\ell)}, i \in \{1, ..., 2^k\}$. We will use this gradient vector to update the free parameters as follows:

$$w_i^{(\ell+1)} = w_i^{(\ell)} - \eta \frac{\partial I(\mathbf{w}^{(\ell)})}{\partial w_i}, i \in \{1, ..., 2^k\} \tag{7.6}$$

where $0 < \eta \leq 1$ is a learning rate parameter.

Let us find a gradient vector of the cost function. Applying the chain rule we can find partial derivatives:

$$\frac{\partial I(\mathbf{w}^{(\ell)})}{\partial w_i} = \frac{\partial I(\mathbf{w}^{(\ell)})}{\partial \varepsilon^{(\ell)}} \frac{\partial \varepsilon^{(\ell)}}{\partial y^{(\ell)}} \frac{\partial y^{(\ell)}}{\partial w_i} \tag{7.7}$$

for each $i \in \{1, ..., 2^k\}$. Each term of (7.7) can be found as follows:

$$\frac{\partial I(\mathbf{w}^{(\ell)})}{\partial \varepsilon^{(\ell)}} = \varepsilon^{(\ell)} \tag{7.8}$$

$$\frac{\partial \varepsilon^{(\ell)}}{\partial y^{(\ell)}} = 1 \tag{7.9}$$

$$\frac{\partial y^{(\ell)}}{\partial w_i} = \frac{1}{M} \sum_{m=1}^{M} \frac{\partial \varphi(x_m^{(\ell)}(T))}{\partial x} \frac{\partial x_m^{(\ell)}(T)}{\partial w_i} \tag{7.10}$$

where

$$\frac{\partial \varphi(x_m^{(\ell)}(T))}{\partial x} = \frac{\sigma \exp(-\sigma(x_m^{(\ell)}(T) - 0.5))}{[1 + \exp(-\sigma(x_m^{(\ell)}(T) - 0.5))]^2} \tag{7.11}$$

We will denote by $u_m^{(\ell)}(t)$ the neighborhood configuration of a cell x_m after transition t. Each $x_m^{(\ell)}(T)$ can be represented as:

$$x_m^{(\ell)}(T) = f(u_m^{(\ell)}(T-1)) = f(x_{m_1}^{(\ell)}(T-1), ..., x_{m_k}^{(\ell)}(T-1)) \tag{7.12}$$

where $x_{m_j}^{(\ell)}(T-1), j \in \{1, ..., k\}$ are the values of those cells that belong to the neighborhood of the cell x_m after transition $T-1$. Applying the chain rule again we can find:

$$\frac{\partial x_m^{(\ell)}(T)}{\partial w_i} = \sum_{j=1}^{k} \frac{\partial f(u_m^{(\ell)}(T-1))}{\partial x_{m_j}} \frac{\partial x_{m_j}^{(\ell)}(T-1)}{\partial w_i} \tag{7.13}$$

where $\frac{\partial x_{m_j}^{(\ell)}(T-1)}{\partial w_i}$ can be found by analogy to (7.13). In this way, we obtained a recursive function:

$$\frac{\partial x_m^{(\ell)}(T-t)}{\partial w_i} = \sum_{j=1}^{k} \frac{\partial f(u_m^{(\ell)}(T-t-1))}{\partial x_{m_j}} \frac{\partial x_{m_j}^{(\ell)}(T-t-1)}{\partial w_i} \tag{7.14}$$

for $0 \leq t < T-1, m \in \{1, ..., M\}$.

Let us consider the base case of this recursion ($t = T-1$):

$$\frac{\partial x_m^{(\ell)}(1)}{\partial w_i} = \frac{\partial f(u_m^{(\ell)}(0))}{\partial w_i} \tag{7.15}$$

Let us define sets P_f and $P_f^{(s)}$:

$$P_f(u_m^{(\ell)}(t)) = \{\min(w_i^{(\ell)}, \mu_{\alpha_i}(u_m^{(\ell)}(t))), i = 1, ..., 2^k\} \tag{7.16}$$

where μ_{α_i} is defined in (6.1) and

$$P_f^{(s)}(u_m^{(\ell)}(t)) = P_f \setminus \{\min(w_s^{(\ell)}, \mu_{\alpha_s}(u_m^{(\ell)}(t)))\}, s \in \{1, ..., 2^k\} \tag{7.17}$$

Then, using this notation, the local transition function f can be represented as follows:

$$f(u_m^{(\ell)}(t)) = \max[P_f(u_m^{(\ell)}(t))] = \max[\max[P_f^{(i)}(u_m^{(\ell)}(t))], \min[w_i^{(\ell)}, \mu_{\alpha_i}(u_m^{(\ell)}(t))]]$$

for some $i \in \{1, ...2^k\}$. Then, applying the chain rule again we obtain:

$$\frac{\partial f(u(x_m^{(\ell)}(0)))}{\partial w_i} = \frac{\frac{\partial \max[\max[P_f^{(i)}(u_m^{(\ell)}(0))], \min[w_i^{(\ell)}, \mu_{\alpha_i}(u_m^{(\ell)}(0))]]}{\partial \min}}{\frac{\partial \min[w_i^{(\ell)}, \mu_{\alpha_i}(u_m^{(\ell)}(0))]}{\partial w_i}} \times \tag{7.18}$$

We can find partial derivatives for the min and max operators, using their representations (3.3) and (3.2). However, in order to do this, we have to define additionally the derivative of the absolute value function $|x|$ at the point $x = 0$. We will consider it to be equal to 0. Taking this into account, we can find partial derivatives as follows:

$$\frac{\partial \max(a, b)}{\partial b} = \frac{\partial \min(a, b)}{\partial a} = \frac{1 - sgn(a - b)}{2} \tag{7.19}$$

where

$$sgn(x) = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x = 0 \\ -1 & \text{for } x < 0 \end{cases} \tag{7.20}$$

Thus,

$$\begin{aligned} \frac{\partial f(u_m^{(\ell)}(0))}{\partial w_i} = & \frac{1}{4}\{1 - sgn(\max[P_f^{(i)}(u_m^{(\ell)}(0))] - \min[w_i^{(\ell)}, \mu_{\alpha_i}(u_m^{(\ell)}(0))])\} \times \\ & \{1 - sgn(w_i^{(\ell)} - \mu_{\alpha_i}(u_m^{(\ell)}(0)))\} \end{aligned} \tag{7.21}$$

It remains to find $\frac{\partial f(u_m^{(\ell)}(T-t-1))}{\partial x_{m_j}}, j \in \{1, ..., k\}$ in (7.13). Although it is possible to find this derivative analytically, it would take a relatively large amount of computation to find it exactly by analytical formula. In order to reduce computational demands of the algorithm, this derivative can be approximated by finite difference. This will, of course, introduce some numerical error, but we consider it to be tolerable in this context. Let us denote

$$f([u_m^{(\ell)}(t)]_{\Delta_j}) = f(x_{m_1}^{(\ell)}(t), ..., x_{m_j}^{(\ell)}(t) + \Delta h, ..., x_{m_k}^{(\ell)}(t))$$

Using this notation, we can approximate the partial derivative as follows:

$$\frac{\partial f(u_m^{(\ell)}(T-t-1))}{\partial x_{m_j}} = \frac{f([u_m^{(\ell)}(T-t-1)]_{\Delta_j}) - f(u_m^{(\ell)}(T-t-1)}{\Delta h} \tag{7.22}$$

Both partial derivatives in (7.14) have been found and the gradient vector of the cost function can be calculated. This completes the description of an iterative step of the learning algorithm.

7.4 Stopping Criterion.

The learning algorithm described in the previous section has not been shown to converge in general. Thus, some criteria for stopping its operations should be defined. Since the suggested learning algorithm is essentially a minimization algorithm applied to the cost function, a reasonable stopping criterion can be based on some unique properties of a local or global minimum of a function. It is important to notice that the cost function (7.5) used in the pattern-by-pattern training mode accounts for the error of the current training example only. To get an indication of the general performance of a learning system, one should take into account the errors of the system for the entire training set and strive for a minimum of the function that does just this. We will define a generalized cost function (or mean-square-error):

$$I_{av}(\mathbf{w}) = \frac{1}{2L}\sum_{\ell=1}^{L}[\varepsilon^{(\ell)}]^2 \tag{7.23}$$

and use it to formulate a stopping criterion for the learning algorithm.

A necessary condition for a point to be a minimum of some function is that the gradient vector of the function at this point be zero. Based on this property, a stopping criterion for the backpropagation algorithm was suggested in [40]: the algorithm is considered to have converged when the Euclidean norm of the gradient vector of (7.23) reaches a sufficiently small threshold. The drawback of this criterion is that for successful trials, learning times may be very long [32].

Another property of a minimum that can be used is that a function is stationary at a minimum point. A criterion based on this property can be formulated as follows: the algorithm is considered to have converged when the absolute rate of change in the generalized cost function per epoch is sufficiently small. A variation of this criterion is to require that the value of the generalized cost function be equal or less than a sufficiently small threshold.

A hybrid criterion was suggested in [40]: the algorithm terminates with parameter vector $\mathbf{w}_{\text{final}}$ when either $\|\mathbf{g}(\mathbf{w}_{\text{final}})\| \leq \epsilon$, where $\mathbf{g}(\mathbf{w})$ is the gradient vector of the generalized cost function, or $I_{av}(\mathbf{w}_{\text{final}}) \leq \tau$, where τ and ϵ are sufficiently small thresholds. This stopping criterion is used in our implementation and is checked after each learning epoch.

Chapter 8

Encoding Real Values into CA Configurations.

A cellular automaton by nature is a discrete system, which performs its computation in space and time which are both discrete. This feature of cellular automata, while being useful and naturally applicable to many problems, raises questions about their applicability to problems which involve continuous computation and especially those that do not have an intrinsic feature of distributed space. Obviously continuous function identification is such a problem.

Though an intuitive reaction may suggest that cellular automata are not appropriate to carry out continuous computations, a discussion in [67] by Toffoli presents a more optimistic view of this problem. According to it, a notion of continuous computation can be introduced even in a classical binary cellular automaton. One can consider the *mean density* δ_V over a certain volume, i.e. the fraction of cells in that volume that are in state "1". While the states of the cells are discrete and binary, δ will always be some number between 0 and 1. As the volume increases, the values of δ will move up and down the unit interval in a smoother and smoother fashion, so that in the limit one can speak of a continuous function.

We will adapt this idea that an information about a real value can be represented by a collection of CA cells, where the precision of this information depends on the size of this collection. We will base our encoding scheme on this idea.

Being a distributed system, a cellular automaton represent and process information in a distributed manner, whereby cells share their knowledge amongst each other during the CA evolution. The result produced by the cellular automaton represents a collective decision of all cells, which they made on the basis of common but distributed knowledge. In a deterministic uniform cellular automaton, which we consider in this study, all cells obey the same rules of state transitions and, in the context of our problem of function identification, all cells work collectively on one global problem. In this case, it would be reasonable to ensure that at the beginning of the CA evolution, all cells possess the information about an input of approximately the same quality. One may argue that information would be propagated to all cells evenly in any case after a certain number of CA transitions. But we suppose that a "fair" initial presentation of initial knowledge to all cells can save the time needed for propagation of this initial information throughout the CA lattice and the automaton can immediately start collective problem solving.

The information, immediately available to a given cell, is that which is present in a neighborhood of this cell. Thus, we would consider a method of input encoding to be "fair" if every neighborhood of the CA lattice represents knowledge of about the same quality. Taking this into consideration, let us introduce one of the possible methods of encoding a real value into a configuration of a 1-dimensional cellular automaton. We will discuss this method first in the context of classical CA with binary states ($Q = \{0, 1\}$) and then we will generalize it for an application to fuzzy cellular automata.

As assumed previously, the input values v lie in the unit interval $[0, 1]$. We will initialize cells' states such that the portion of cells initialized to state "1" in every given neighborhood on $\mathcal{L}$ is as close to v as possible. One can roughly implement

this by using the following rule:

$$x_i = \begin{cases} 1, \text{ if } (i \bmod k) < \lfloor kv \rfloor \\ 0, \text{ otherwise} \end{cases} \tag{8.1}$$

where v is an input value, and k is the size of the neighborhood.

Due to the fact that we use the floor operator - $\lfloor kv \rfloor$ - the overall portion of ones in the CA initial configuration may appear to be less than v. In this case we can do a little refinement. The number of ones that still should be added can be found as follows:

$$b = \frac{M}{k}(kv - \lfloor kv \rfloor) \tag{8.2}$$

Then this number of additional ones should be evenly distributed on $\mathcal{L}$ by flipping every $\frac{z}{b}$ zero state on $\mathcal{L}$. z is a number of cells initialized to zero state after the rough method (by (8.1)) and it is equal to:

$$z = \frac{M}{k}(k - \lfloor (kv) \rfloor) \tag{8.3}$$

Thus by (8.2) and (8.3) every $\frac{k-\lfloor (kv) \rfloor}{kv-\lfloor (kv) \rfloor}$ zero state in the initial configuration of the cellular automaton should be changed to state "1".

Similar method can be used to encode input values into initial configurations of fuzzy CA. In this case, cells that should be initialized to one, according to the above algorithm, will be assigned a random value $x_i \in (\frac{1}{2} + \epsilon, 1]$ and those that should be assigned zero - a random value $x_i \in [0, \frac{1}{2} - \epsilon]$, with some value ϵ to avoid ambiguity at $\frac{1}{2}$.

Clearly, the encoding accuracy in this scheme is dependent on the neighborhood size: larger neighborhood sizes permit more accurate encoding. At the same time, an increase of the neighborhood size makes local computations of a cellular automaton more intense and complex. Also, what is even more important, a number of free parameters, that have to be learned, increases exponentially. This makes a solution space larger which, although can reduce the model's bias, makes the learning task

more complex and requires more computational time. A fixed neighborhood size imposes a limit on the encoding accuracy and therefore on the function approximation accuracy.

It should also be noted that not the entire space of the possible cellular automaton's configurations is employed in the encoding, that is a mapping of real values onto a set of cellular automaton's configurations is not surjective. In this sense, our encoding scheme does not explore all the potential power of the cellular automata structure. Nevertheless it possesses other qualities (discussed earlier in this chapter) which, we believe, are important if a uniform model of cellular automata is meant to compute continuous mappings.

Chapter 9

Experiments and Results.

9.1 Objectives of the Experiments.

To the best of our understanding, this study is the first attempt to use cellular automata in the context of function identification from a set of input-output pairs. Thus far, no learning algorithms were developed for this purpose. Even the capacity of cellular automata to compute and represent knowledge about continuous functions has never been previously assessed. The present work can be considered as a feasibility study and because of this reason we nave decided to start with a relatively simple task - identification of univariate continuous functions.

As was pointed out initially, an absolutely general solution strategy to the problem of function identification has not yet been found. All approaches used today be they in statistics, applied mathematics or artificial intelligence have limitations. These limitations depend on the intrinsic capabilities of the models employed in each method to represent certain functions, and none of the existing methods can be claimed to be good or bad *in general* [17]. Based on this point of view, we set out our main objectives to validate an ability of cellular automata to perform complex continuous computation, study the behavior of a new learning system and compare

its efficiency in approximating different types of continuous functions. We focused our analysis of a new learning approach, presented in this work, on the following:

1. The ability of the proposed algorithm to perform a learning task and make an underlying fuzzy cellular automata model generalize knowledge represented in a set of training examples.

2. The effect of the parameters, that determine a structure of a cellular automaton, on the performance of a learning system.

3. The effect of a learning rate parameter η in (7.6) on the algorithm's performance.

4. The efficiency of a fuzzy cellular automata model in approximating continuous functions of different types.

Performance of a fuzzy cellular automaton learning system was studied by means of experiments. In the rest of this section we will present the methodology employed in our experiments together with the results and observations obtained from these experiments with respect to the objectives stated above.

9.2 Experimental Designs and Discussion of the Results.

As mentioned in Section 9.1, there is no analysis available regarding the ability of a cellular automaton based model to represent continuous functions. Consequently, the first step was to verify the correctness and ability of a new learning algorithm to perform supervised learning. To distinguish the aspects of learning and knowledge representation capabilities, we first tested our algorithm using test functions computed by another cellular automaton with known values of free parameters. In

this case we could be sure that a fuzzy cellular automaton was capable of computing these functions and performance of the learning algorithm itself could be validated. A set of experiments were conducted in which both a fuzzy cellular automaton, that generated a test function (target FCA), and a learning FCA had the same size of a cellular lattice, M, neighborhood size, k, and a number of transitions T which both automata were allowed to perform before an output value was decoded from a cellular automaton configuration.

In the case of this test, the learning task appeared to be quite easy for a fuzzy cellular automaton system. It was consistently able to find a satisfactory approximation of a target function (approximation with a value of mean-square-error (MSE) over the entire training set less that 0.001). The average number of learning epochs that it took a learning FCA to identify a target function was five with a number of training examples presented at each epoch equal to 40 (see Table 9.1). Learning examples in this and all other tests were uniformly distributed over the real unit interval $[0, 1]$. Since computational capabilities of fuzzy cellular automata are not an issue in this context, one can see that the learning algorithm itself is functional and fast. An example of a test function generated by a fuzzy cellular automaton and its approximation can be found in Figure 9.1.

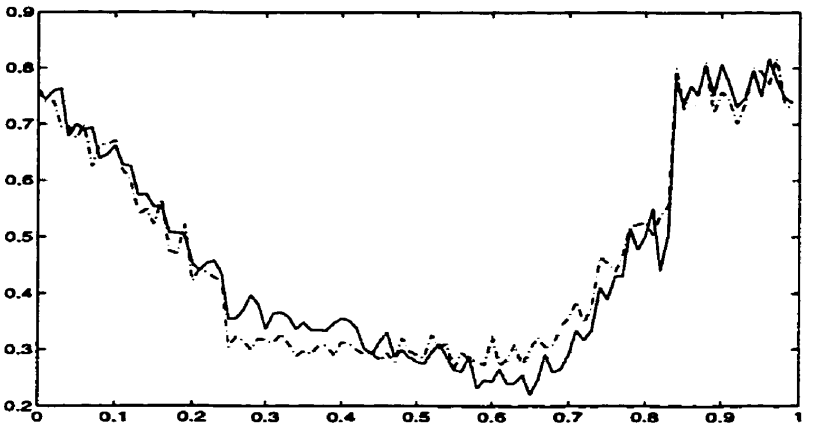
Table 9.1: Test1.

Exper. #	M	k	T	# epochs	MSE
1	65	3	1	3	0.0006
2	100	5	1	6	0.0006
3	65	3	2	9	0.0009
4	65	3	3	2	0.0004

Target and learning FCA have the same values of parameters.

The first attempt of understanding the influence of such parameters as neighborhood size, k, and number of allowed transitions, T, on the computational capabilities

Figure 9.1: FCA Generated Test Function ".–.–." and Its Approximation"—".



Parameters of both target and learning FCA are the same: $M = 65; k = 3; T = 1$.

of fuzzy cellular automata was done through the two following tests.

In Test 2, lattice and neighborhood sizes of a target and learning FCA were set to be equal, but a number of transitions T - different. First, a value of the parameter T of a target FCA was set to 1 and that of a learning FCA varied from 2 to 5 (see Table 9.2). In this test, only a learning FCA with $T = 2$ was able to find a satisfactory approximation. In all other cases, the learning algorithm did not converge and the resulting models had a very low accuracy. This outcome suggests that with an increase of the T parameter, representational/computational capabilities of fuzzy cellular automata decrease. When we mention computational capabilities of FCA, we refer only to those that are relevant and specific in the context of our problem, that is an FCA ability to compute complex continuous functions. Indeed, graphs of functions computed by FCA with $T > 2$ show that these functions are almost linear and more than that they tend to be close to constant functions. A range of these functions does not exceed an interval $[0.4, 0.6]$ most of the time. If one observes time evolution of such a fuzzy cellular automaton, one immediately notices that there are just a few (2 to 4) distinct cell states present in a cellular automaton configuration after transition 3. In addition the cellular automaton reaches a steady point in its

evolution or exhibits a shift-like behavior after a small number of transitions.

Table 9.2: Test 2.

Exper. #	M	k	T	# epochs	MSE
1	65	3	2	5	0.0009
2	65	3	3	did not converge	0.0164
3	65	3	4	did not converge	0.0230
4	65	3	5	did not converge	0.0571

Target and learning FCA have the same values of parameters M and k.

$T = 1$ for a target FCA.

A few more experiments were conducted, in which the value T of a target FCA was set to 3 and that of a learning FCA varied from 1 to 5 (see Table 9.3). In all these cases, fuzzy cellular automata found successful approximations. This supports the conclusion that fuzzy cellular automata with smaller values of T (especially less than 3) have stronger computational powers. Starting from $T = 3$, additional increases of T do not significantly alter the situation any further.

Table 9.3: Test 2 (continued).

Exper. #	M	k	T	# epochs	MSE
5	65	3	1	4	0.0006
6	65	3	2	10	0.0005
7	65	3	3	4	0.0009
8	65	3	4	24	0.0009

Target and learning FCA have the same values of parameters M and k.

$T = 3$ for a target FCA.

In Test 3, a neighborhood size was set to be different for the target and learning

FCA with the other parameters being equal. Neighborhood size of the target FCA was set to 6 and that of the learning FCA varied from 3 to 8. Two series of such experiments were conducted, one with the value of T equal to 1 and the other with $T = 3$. This test demonstrated that the neighborhood size also affects computational capabilities of fuzzy cellular automata (see Table 9.4). A learning FCA with a size of a neighborhood smaller than that of a target FCA ($k < 6$), was not able to find a satisfactory approximation. In the case of $T = 3$, a learning FCA with the neighborhood size equal to 5 (still smaller than that of a target FCA) was however able find a satisfactory model. According to the conclusions of Test 2, a test function, computed by a target FCA with $T = 3$, was simpler than in the case $T = 1$ and the neighborhood size probably had a less significant effect.

Table 9.4: Test 3.

Exper. #	M	k	T	# epochs	MSE
1	65	3	1	72	0.0088
2	65	5	1	17	0.0016
3	65	7	1	13	0.0009
4	65	8	1	10	0.0009
1	65	3	3	24	0.0103
2	65	5	3	8	0.0009
3	65	7	3	5	0.0008
4	65	8	3	6	0.0008

Target and learning FCA have the same values of parameters M and T.
$k = 6$ for a target FCA.

Thus, computational capabilities of fuzzy cellular automata seem to increase with an increase of the neighborhood size. It was also presumed that size of a cellular automaton lattice, M, can affect the performance of a learning system. It was also noticed that a learning rate parameter η in (7.6) had some influence on

the properties of the learning algorithm. To examine these effects, we employed a statistical technique - response surface modeling with regression analysis. There were two responses of the learning system that we were interested in: speed of convergence and accuracy of obtained function approximations. A number of learning epochs served as an indicator of the convergence speed, and mean-square-error (7.23) of a final model identified by a fuzzy cellular automaton - as an indicator of the model's accuracy. Parameters M, k and η were considered to be the factors influencing response variables. A function that represents a relationship between these factors and a response variable is called a response function. Naturally we do not know a true form of this function and have to model it by some means. For this purpose we used regression analysis from statistics. It was assumed that a response function in our case was more complex than a linear one, and we decided to use a quadratic model of a response surface:

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i + \sum_{i=1}^{p} \beta_{ii} X_i^2 + \sum_{i=1}^{p} \sum_{j=1}^{p} \beta_{ij} X_i X_j \qquad (9.1)$$

where $X_1, X_2, ..., X_p$ are the influencing factors, Y is a response variable and $\beta_0, \beta_i, \beta_{ii}, \beta_{ij}$, $i, j \in \{1, 2, ..., p\}$ are the unknown parameters that we will have to estimate.

A design of experiments, by which the observed values of a response are collected for estimating the parameters in a second-order model (9.1), is called a second order design. There are several different methodologies of the second-order designs. In our study we used Central Composite Design with orthogonal and uniform precision properties [36]. According to this methodology, a set of the parameter values combinations is designed and experiments are conducted using these combinations. After the response values are collected from these experiments, regression analysis is applied in order to estimate the parameters of (9.1). We performed two sets of such experiments to obtain the surface models for a speed of convergence response and two sets - for an accuracy response variable. In the early stages of working with a new learning system, it was noticed that some target functions were easier for fuzzy cellular automata to learn than others. For some functions, the learning algorithm always managed to find satisfactory models. We selected two functions

such as these and used them as the test functions in the experiments designed for modeling of the convergence speed response (Tests 4 and 5). For two other sets of experiments (Tests 6 and 7), where an objective was to model the approximation accuracy, more difficult functions, from fuzzy cellular automata point of view, were used. In these cases, the learning algorithm often converged at some local minimum of the error surface, producing the approximations with different levels of accuracy.

Experimental settings and results are listed in Table 9.5. In all further experiments, cellular automata with $T = 1$ were used, since we expected the most sophisticated computational capabilities from such a setting. These data ware processed with the "Design-Expert" software package, which performed regression analysis and produced quadratic models of the response surfaces. It should also be noted that there were several experimental outcomes that were not well predicted by the response surface models and they seemed to significantly differ from the overall behavior of the learning system. This suggests that response functions may have even more complicated forms than a quadratic one, and for specific target functions there may be some especially good or bad combinations of parameter values. But these cases are rare and general trends in the behavior of the learning system can still be discovered. The obtained quadratic response surface models explain between 60 and 80 percent of the response value variations caused by the studied factors M, k and η. The remaining portion of the variations can also be attributed to the random factors, present in the learning process. One of them is a random initiation of the free parameters. The performance of a learning system can depend on how close an initial operating point (initial values of free parameters) happens to be to a good minimum of the error surface. The fact that the training examples are presented to a learning system in a different (random) order at each learning epoch, makes a walk of the free parameters' values stochastic in a parameters' space. This can additionally affect the learning system's performance which may not be adequately captured by a relatively simple and deterministic model of a response surface. How-

Table 9.5: Tests 4-7. Experimental Settings and Results.

Exper. #	M	k	η	Number of epochs		MSE	
				Test 1	Test 2	Test 3	Test 4
				Y_1	Y_2	Y_3	Y_4
1	65	3	0.2	341	9	0.0075	0.0023
2	65	3	0.5	1000	2	0.0070	0.0041
3	65	7	0.2	19	41	0.0010	0.0009
4	65	7	0.5	41	23	0.0011	0.0008
5	65	3	0.2	60	12	0.0073	0.0027
6	150	3	0.5	1000	3	0.0077	0.0087
7	150	7	0.2	36	37	0.0009	0.0008
8	150	7	0.5	30	4	0.0009	0.0009
9	150	5	0.1	114	32	0.0010	0.0014
10	108	5	0.6	24	10	0.0015	0.0009
11	108	2	0.35	1000	30	0.0036	0.0051
12	108	10	0.35	23	15	0.0009	0.0009
13	108	5	0.35	10	12	0.0012	0.0007
14	36	5	0.35	17	10	0.0008	0.0013
15	178	5	0.35	16	13	0.0009	0.0008
16	108	5	0.35	14	12	0.0009	0.0009
17	108	5	0.35	18	16	0.0012	0.0016
18	108	5	0.35	42	22	0.0008	0.0011
19	108	5	0.35	23	25	0.0008	0.0015
20	108	5	0.35	13	9	0.0016	0.0008

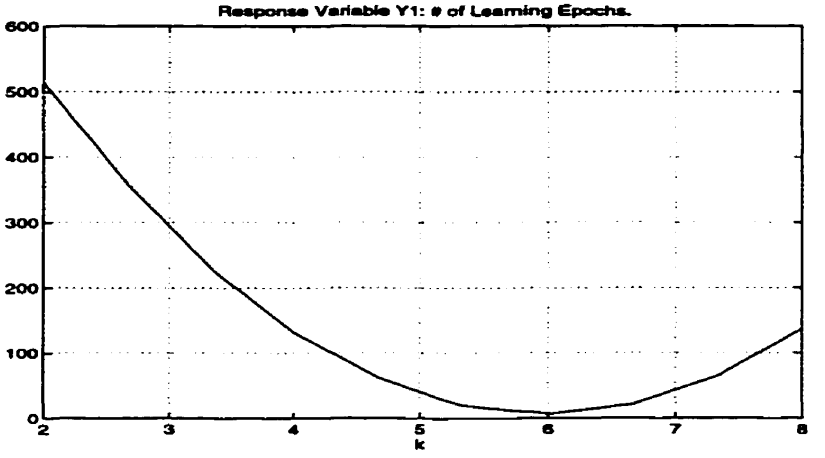
ever, the obtained response surface models can provide us with a reasonable insight into the influence of the learning system's parameters on the learning process.

In a case of a Test 4, where the speed of convergence was of interest, a model of

the response surface appeared to depend only on the neighborhood size parameter, k:

$$Y_1 = 1151.6 - 382.8555k + 32k^2 \tag{9.2}$$

Figure 9.2: Results of Test 4.



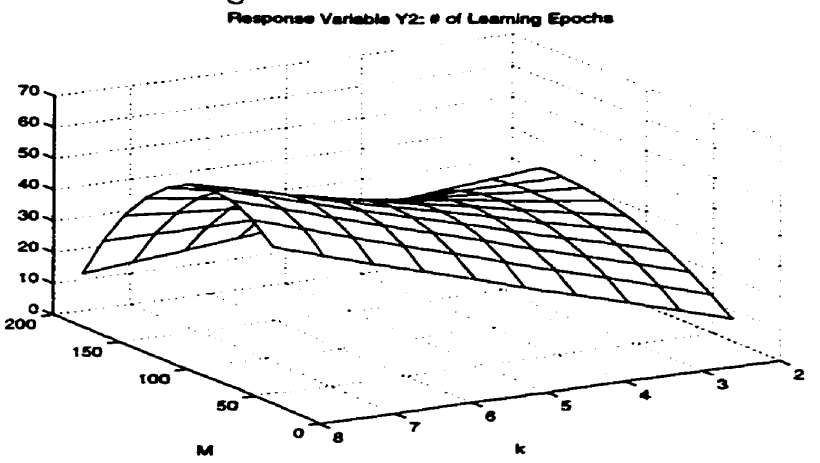
As can be seen from Figure 9.2 , convergence speed increased as the neighborhood size increased up to $k = 6$ and then started to decrease again. The reason for this, in our opinion, is that as k increases , the computational capabilities of a learning FCA increase and a learning system can learn better and faster. However, with an increase of a neighborhood size, the complexity of the learning task increases (since a number of free parameters to be learned increases and is equal to 2^k) which requires more training. Because of this, after a certain point (in this case $k = 6$) a further increase of a neighborhood size contributes more to the learning complexity than to the FCA computational capabilities.

A model of a response function for the Test 5 shows that a speed of convergence is influenced by all three factors (M, k, η):

$$Y_2 = 2.00384 - 19.07336\eta + 0.3M + 5.75731k + 10^{-5}M^2 + 0.0613Mk - 0.006M^2k \tag{9.3}$$

Figure 9.3 depicts a dependence of the convergence speed on the values of parameters k and M when η is fixed to 0.35. In this case, we can see that the best performance is achieved for the small sizes of a neighborhood and cellular automata

Figure 9.3: Results of Test 5.



lattice, which is somewhat contradictory to the results of the previous test. But this can be explained by the fact that the test function used in this experiment is much easier for the fuzzy cellular automaton than the one used in a Test 4. Because of this, the learning FCA can approximate this target function with a small neighborhood. As was pointed out previously, the learning task is less complex in the case of a smaller neighborhood size and requires less learning epochs. A tendency of increasing the convergence speed with a decrease of the neighborhood size shows up, however, only for small and medium lattice sizes. As a number of cells increases further, a learning system benefits from the increase of a neighborhood size and a good performance can be obtained again with large neighborhoods and lattices. An effect of the lattice size's change on the system's performance is rather small for small neighborhoods, but becomes more significant as a neighborhood size increases. In this test, fuzzy cellular automata seem to favor the combinations where either both sizes are small, or both - large. Probably a certain ratio of two sizes provides the best performance of an FCA system.

Figures 9.4 and 9.5 illustrate an effect of a learning rate parameter η on the convergence speed. In this case, for all values of k and M, the convergence speed improves as η increases.

Response variables in the tests 6 and 7 represent an accuracy of a final approxi-

Figure 9.4: Results of Test 5.

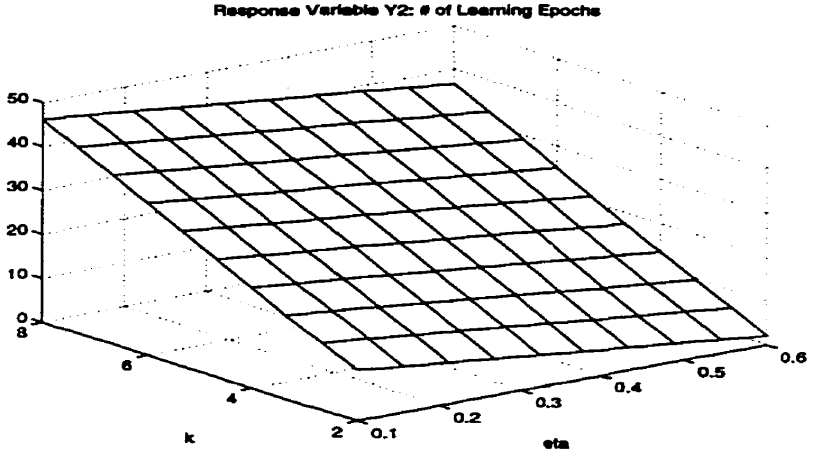
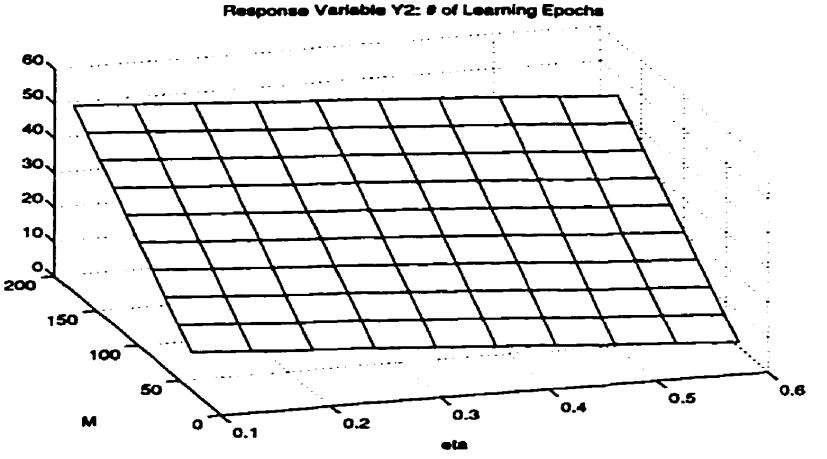


Figure 9.5: Results of Test 5.

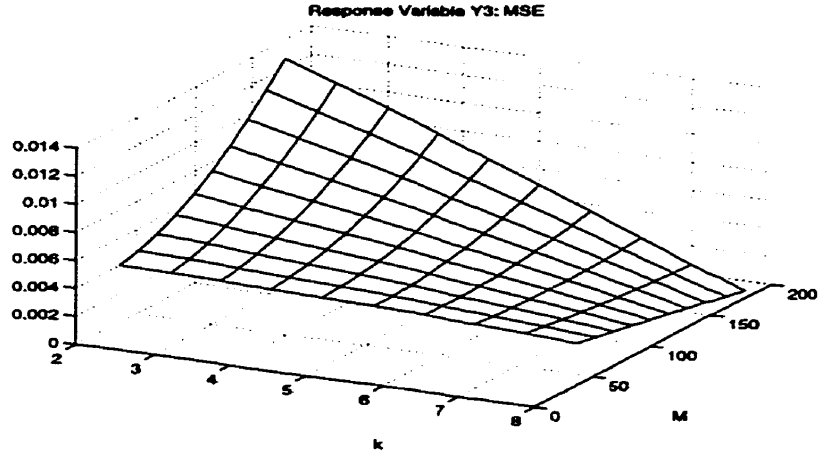


mation produced by a learning system. The model of the response surface, induced from the experimental data of the Test 6, shows that only factors k and M influence the response variable:

$$Y_3 = 10^{-6}(3500 + 184k + M + 0.4M^2 - 22k^2 - 0.19Mk - 0.051M^2k) \qquad (9.4)$$

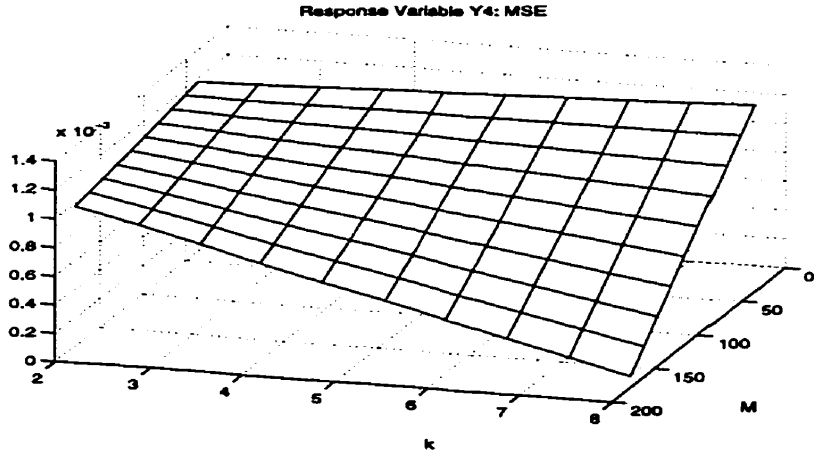
Figure 9.6 illustrates that the accuracy of an identified model of a target function improves with an increase of the neighborhood size. This observation is consistent with the previous conclusions that an increase of the neighborhood size makes fuzzy cellular automata capable of more complex computations. This effect becomes more significant as a lattice size increases. The magnitude of an effect of the parameter M

Figure 9.6: Results of Test 6.



on the accuracy response increases with the decrease of the neighborhood size and for small values of k, large values of M have a negative effect on the approximation accuracy.

Figure 9.7: Results of Test 7.



A model of a response function for the Test 7 shows that the parameter η influences an accuracy of approximation as well:

$$Y_4 = 10^{-5}(110+0.1M+6.35k+1000\eta+0.0001k^2-140k\eta+8.61M\eta-0.12Mk) \quad (9.5)$$

The results presented in both Figures 9.7 and 9.8 support the conclusion that an increase of the neighborhood size has a positive effect on the accuracy of approximations. With respect to the parameter M, one can make certain observations from

Figure 9.8: Results of test 7.

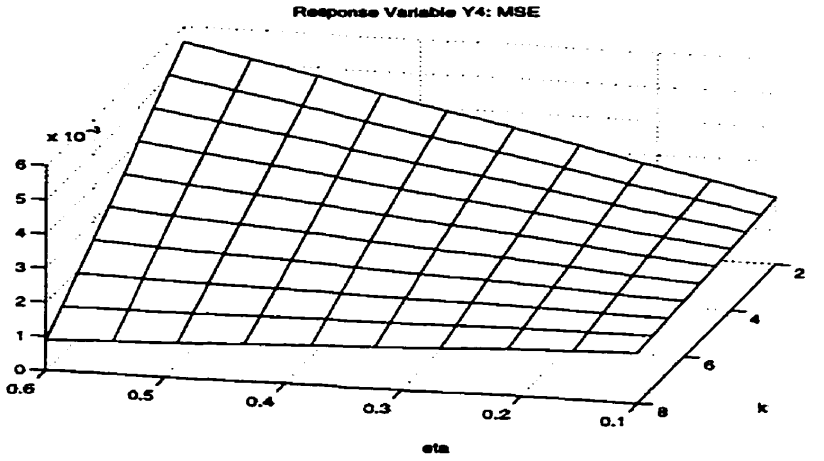


Figure 9.7 which are not absolutely consistent with the conclusions of the previous test regarding this parameter. But in this test, a target function was generated by a known FCA with the same lattice size as that of a learning FCA. In this case, it was a neighborhood size that played a more important role. Increasing the lattice size, probably did not add any significant additional complexity to a target function, but was somewhat helpful to the learning FCA for all values of k.

Figure 9.8 illustrates that the learning rate parameter, η, has different effects on the accuracy depending on the neighborhood size. For small neighborhoods, an increase of η has a significant negative effects on the response value, whereas for the large neighborhood sizes, performance of the algorithm is less sensitive to the values of η but improves with the increase of η.

A fuzzy cellular automaton learning system was tested on a variety of continuous functions in order to assess its efficiency in learning different kinds of functions. Table 9.6 lists the test functions along with the values of MSE which were achieved during the experiments. (Note that approximation was considered to be satisfactory when $MSE < 0.001$ and the algorithm terminated if this condition was met.) Table 9.6 represents the results in the order of ascending MSE values. Figures that illustrate graphs of the test functions together with their corresponding approximations can be found in the appendix.

Table 9.6: Real Valued Functions Used in the Experiments.

Function Type	Function	k	MSE
Polynomial	$y(x) = 1.5x^2 - 1.35x + 0.75$	5	0.0001
Exponential	$y(x) = \frac{1}{150}\exp(0.5x^2)\exp(0.8(x-2)(x-3))$	5	0.0001
Unimodal	$y(x) = \frac{2}{3}\exp(x\sin(5x))$	5	0.0002
Polynomial	$y(x) = 2x^3 - 0.8x^2 - 0.8x + 0.6$	5	0.0004
Polynomial	$y(x) = 2x^4 - 0.8x^3 - 0.8x^2 + 0.2x + 0.2$	5	0.0004
Bimodal	$y(x) = \frac{1}{3}\exp(x\sin(10x))$	5	0.0005
Bimodal	$y(x) = \frac{1}{5}\sin(10x) + 3x^2 + 2x$	5	0.0006
Low-frequency Harmonic	$y(x) = \frac{1}{2}\sin(5x) + 0.5$	7	0.0009
High-frequency Harmonic	$y(x) = \frac{1}{300}(\sin(30x)\exp(5x) + \sin(x)) + 0.5$	8	0.0027
High-frequency Harmonic	$y(x) = \frac{1}{8}\sin(13x)(3 + sin(3x+1))$	8	0.0036
High-frequency Harmonic	$y(x) = \frac{1}{8}(1 + \sin(15x))(3 + sin(7x+1))$	8	0.0051
Highly Non-smooth	$y(x) = \frac{1}{3}\sin(10x^2 + \cos(25x)) + 0.5$	8	0.0055

As we can see from Table 9.6, the learning system was most successful with such target functions as polynomials, exponential, bimodal, unimodal and law-frequency harmonic functions. Next on the scale was a harmonic function with an increasing frequency and amplitude. The most difficult functions for a fuzzy cellular automaton system were high-frequency harmonic and highly non-smooth functions. It was also noticed that for "easier" functions, the learning algorithm converged faster and good approximations could be found with the smaller neighborhood sizes.

These "easier" functions have a common feature of a relatively high degree of smoothness and a small number of modal points. It appears that a range of the function's values has an impact on the ability of a learning system to identify this given function. When the same test functions were slightly scaled down, the algorithm was able to find more accurate approximations than before scaling.

In many cases, the approximation accuracy worsens at the end of the unit interval - $[0, 1]$ - as compared to the rest of the interval. This problem, as well as the one with an effect of a function values range, from our point of view, can be partially attributed to the imperfection of the encoding/decoding scheme used to represent real valued inputs/outputs on a cellular automata lattice. Accuracy of this encoding decreases within $[0.9, 1]$ portion of the unit interval. Initially, we used simply an average of the cells' states in an output configuration as a decoding function . In this case, the range of the function, computed by fuzzy cellular automata with the random settings of free parameters, did not span outside $[0.03, 0.65]$. In our opinion, part of a reason is in the usage of the standard fuzzy intersection - minimum operator - in a transition rule of a fuzzy cellular automaton. In the case of a fuzzy transition function (6.2), this operator does not favor the cell's values close to 0 or especially 1 and consequently the average of the cell values in an output configuration is not likely to fall into these extremes. To make up for this shortcoming in the context of our application, we used a sigmoid function on the decoding step (7.1). A sigmoid function, close to a threshold, would be able to solve a range problem. On the other hand, the closer a shape of this function is to a threshold, the faster its derivative vanishes to 0. This would complicate and slow down the learning algorithm since the values of the free parameters' updates which are made at each learning iteration depend on the values of this derivative and would vanish to 0 as well. An acceptable choice was to use a sigmoid with a shape that could compromise two problems. This is why a range problem was not solved completely and fuzzy cellular automata found it to be more difficult to approximate those parts of the real functions where functions' values are close to 0 or 1.

Another option would be to force the values assumed by cells at the initial cellular automata configurations, which encode input values, to be closer to 0 and 1. We implemented and tested an encoding scheme which used only binary values as opposed to random values in $[0, 1]$ (see Chapter 8). This approach appeared to be helpful for a function's range problem (see figure A.14). It also has the following

"side-effect".

A function, computed by a fuzzy cellular automaton, where input values are encoded with the arbitrary real numbers in $[0, 1]$, has a rather large number of small local fluctuations (a function zigzags with a small amplitude). When only binary values are used for encoding, FCA-computed functions become smoother (compare figures A.10 and A.11 with figures A.13 and A.14). An application of this encoding scheme does not improve the overall learning abilities of a fuzzy cellular automaton system, however, smoothness of the approximation models is normally a desirable feature[18].

Chapter 10

Conclusion.

This thesis sets out to study a cellular automata model of computation from the perspective of its applicability to the problems involving continuous computations as well as its learning capabilities. This thesis contains background material on cellular automata, fuzzy set theory, learning paradigms and a function identification problem. Existing research related to these areas has also been discussed. In this study, a cellular automata model was used as a tool in the continuous function identification problem. The research community understands very well that cellular automata are extremely difficult to analyze theoretically and often their properties can not be derived or proven by analytical means. Due to this fact, the efficiency of a cellular automata application to the function identification problem, considered in this thesis, was assessed by empirical means. A learning system based on a fuzzy cellular automaton was implemented in software and tested by means of experiments. This work together with the results obtained are summarized as follows.

1. **Cellular automata and continuous computations.**
   Cellular automata represent a model which was originally introduced and used primarily for a discrete type of computations. In the present study they were successfully applied to a new problem involving continuous computations - the

problem of continuous function identification. This study demonstrated that cellular automata are capable of computing complex continuous mappings.

2. **Cellular automata as learning systems.**

   Several studies have already been conducted in an attempt to design cellular automata able to perform some useful predefined tasks when no a priori knowledge about their transition functions was available. This study once more demonstrated that cellular automata models can be used as knowledge representation and acquisition tools, and they can be designed (or trained) for a particular application using various machine learning techniques. We designed a gradient descent type learning algorithm for a cellular automaton based learning system intended for continuous function identification problem.

3. **Fuzzy model of cellular automata.**

   Fuzzy model of cellular automata was introduced which enabled to combine cellular automata property of local logical computations with the requirements of continuity of a state transition function introduced by a gradient based learning approach.

4. **Mapping real values onto cellular automata structure.**

   An encoding scheme was proposed which allowed us to map real values onto a cellular automata discrete structure. The main feature of this encoding scheme consists in that, when an input value is encoded into a cellular automaton initial configuration, each cell has an immediate access (through its neighborhood) to the knowledge of an input value of about the same quality. Yet any cell by itself does not have complete information about an input value and this information is distributed across an entire cellular automaton lattice. The fact that each neighborhood has initial information of about the same quality, in our opinion, provides cellular automata with an opportunity to proceed to the collective problem solving at once, omitting an information propagation phase, which would probably be necessary in the case of uniform cellular automata. It should be noted however that the encoding accuracy depends on

the size of a neighborhood: the larger a neighborhood is, the more accurate encoding is possible. At the same time, an increase of the neighborhood size introduces additional complexity into a learning task. A dependence between the encoding accuracy and complexity of the learning could be an impediment in practical applications where highly accurate approximations are required but computational resources are limited. We intend to look for a solution to this problem in our future work.

5. **Learning algorithm.**

   A proposed learning algorithm is based on a gradient descent optimization technique and was designed specifically for a cellular automata type of computation. If cellular automata were simulated on a parallel machine, the learning algorithm could be efficiently executed in parallel as well. The proposed learning algorithm proved to be functional and capable of making an underlying fuzzy cellular automata model to generalize knowledge present in a training data set. The algorithm was shown to be quite robust with respect to the only "tuning" parameter attributed to the algorithm itself - a learning rate parameter η. There is a very small effect of this parameter on the performance of a learning system:

   - for large neighborhood sizes ($k \geq 5$), an increase of η has a positive effect on the system's performance;
   - in the case of highly non smooth target functions, for small neighborhood sizes an increase of η negatively affects the system's performance.

   However in general, the algorithm converges and can perform an identification task successfully with the default value $\eta = 0.35$.

6. **Effects of cellular automata parameters on the performance of a learning system.**

   An effect of the parameters that define a structure of cellular automata -

lattice size, M, and neighborhood size, k - on the performance of a fuzzy cellular automaton learning system was studied.

- **Neighborhood size.**

  Neighborhood size is the most influencing parameter of the learning system. It has an effect both on the convergence speed of the algorithm and the approximation accuracy of the models, identified by the learning system:

  - the capabilities of fuzzy cellular automata to compute complex continuous mappings increase with an increase of the neighborhood size;
  - an increase of the neighborhood size enables us to obtain more accurate approximations.
  - as the computational abilities of fuzzy cellular automata increase with an increase of the neighborhood size, the learning system can incorporate and gain knowledge faster. But at the same time, as k increases, a number of free parameters that have to be learned increases as well which requires more training. For a given continuous function, that has to be identified, there probably is a threshold where an increase of k adds more to the learning task complexity than to the learning abilities of the system. When accuracy of an approximation is a main goal of the function identification task and a learning time is not a crucial factor, the learning system can benefit from an increase of k.

- **Cellular automaton lattice size.**

  A size of a cellular automaton lattice, M, has a less significant effect on the system's performance, as compared to the effect of the neighborhood size. It is rather a combination of values of these two parameters that has an effect. Cellular automata with small neighborhood sizes do not benefit from large lattice sizes but for large neighborhood sizes, an increase of M has a positive effect. Thus, a value of M should be in some way

proportional to the neighborhood size for the system to perform well.

7. **Approximation abilities of a learning system.**

   The fuzzy cellular automaton learning system was tested on a variety of univariate continuous functions. The system clearly demonstrated an ability to perform a function identification task. It was more efficient in approximating such functions as exponential, polynomials, unimodal, bimodal and low frequency harmonic functions. Highly non smooth and high frequency harmonic functions were more challenging for the learning system. We believe that the difficulty of the fuzzy cellular automaton learning system to accurately approximate highly-non smooth functions should be attributed primarily to the imperfection of an encoding scheme and not to the cellular automata model itself. Limited encoding accuracy makes it more difficult for cellular automata to adapt and react to very rapid and frequent changes of functions' values. We feel that the abilities of cellular automata based systems to perform complex continuous computations and learn from examples depend tremendously on the efficiency of the representation of real values by cellular automata configurations.

8. **No a priori knowledge required for the design.**

   An important feature of a fuzzy cellular automaton learning system is that its structure is rather simple and does not require almost any a priory design decisions. As discussed in section 9.2, a neighborhood size is the most important parameter and its effect on the performance of the learning system is quite straightforward. A suitable value of this parameter can be easily found for a particular target function such that it is large enough to ensure a satisfactory level of approximation accuracy but not too large to exceed the available computational resources.

This study confirmed that powerful features of cellular automata, already demonstrated on the numerous discrete problems, can be successfully employed in the

applications involving continuous computations. A diversity of cellular automata computational abilities can be of a great utility to the problems which combine both continuous and discrete features. A research in the field of function identification problem can be enriched by a new approach, involving a powerful and versatile computational model.

Successful outcomes of this study encourage further research in this field and first of all prompt an application of a fuzzy cellular automaton learning system to more complex identification problems. It would be a good idea to conduct experiments with high-degree polynomials to estimate the relationship between the "smoothness" of the target functions (number of modal points) and the structural requirements of a cellular automaton based learning system.

A cellular automaton learning system should certainly be applied to problems that involve identification of multivariate functions. Such an application demands an encoding scheme for representation of multiple inputs on a cellular structure. Regarding this issue either the encoding scheme proposed in this study has to be generalized for a multiple input case or a new scheme has to be invented. In any case, the problems that were encountered while using the present encoding scheme should be carefully considered and solved. There are two important issues that should be taken into account in this context. First, the dependence of the neighborhood size on the number of inputs should be avoided by any means or reduced to a minimum. The reason for such a requirement is that as the size of the neighborhood increases, the local computations at each site of a cellular lattice become more intense and complex which diminishes one of the most valuable characteristics of cellular automata - simplicity of basic components. Second, an attempt should be made to efficiently use the entire space of the cellular automata state configurations in an encoding mechanism.

The above considerations as well as many others may suggest the use of different models of cellular automata. For example, a non-uniform model may facilitate

applications of some position-based encoding schemes and help to reduce connectivity requirements. It would certainly be very interesting to perform identification of some real-life processes. A large number of such processes are stochastic which suggests an application of a probabilistic model of cellular automata. Many complex processes can be decomposed into sequences of simpler ones. Each of them can be modeled by a cellular automaton based system whereas all these systems can be reunited in the frame of hierarchical cellular automata. As was discussed in Chapter 2, a cellular automata model can accommodate numerous modifications equipping it with many interesting and valuable features which certainly can be useful in many applications.

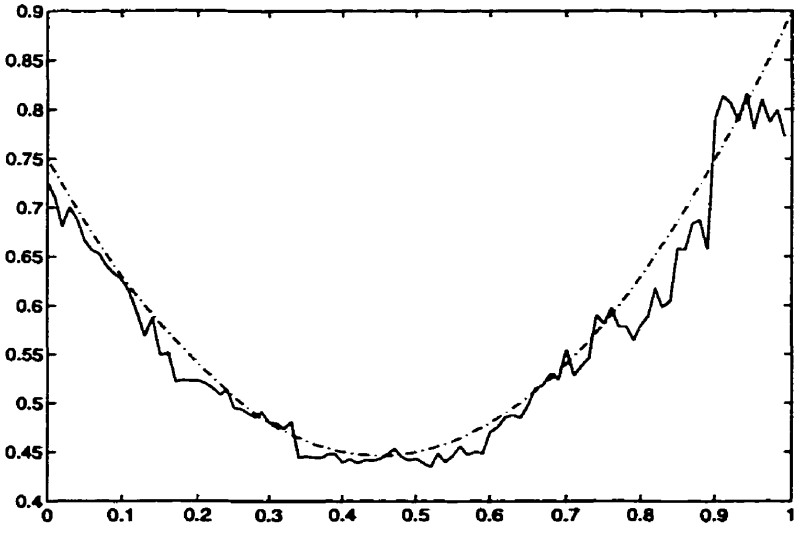
As this study demonstrated, a fuzzy model of cellular automata is an interesting tool that facilitates the introduction of continuous computations and learning capabilities into the cellular automata model. In our opinion it is worthwhile to study this model in more details. The transition function of the fuzzy cellular automaton designed in this work is constructed from standard fuzzy operations. However, as was pointed out in Chapter 3, there are other functions that can play the role of the intersection, union and complement for fuzzy sets. It would be interesting to make use of these operations in the design of a transition function. Amongst the objectives of the new design should certainly be a solution to two problems encountered in this study. One of them is the decreasing capability of fuzzy cellular automata to compute complex continuous mappings as the number of transitions in CA evolution increases. The other one concerns the ability of a fuzzy cellular automaton to compute the values of continuous functions near the borders of the unit interval.

And finally one of the most interesting and challenging tasks would be the design of new learning techniques for the cellular automata model. Techniques already developed for connectionist systems can be adopted to cellular automata and new learning concepts should be sought.

Appendix A

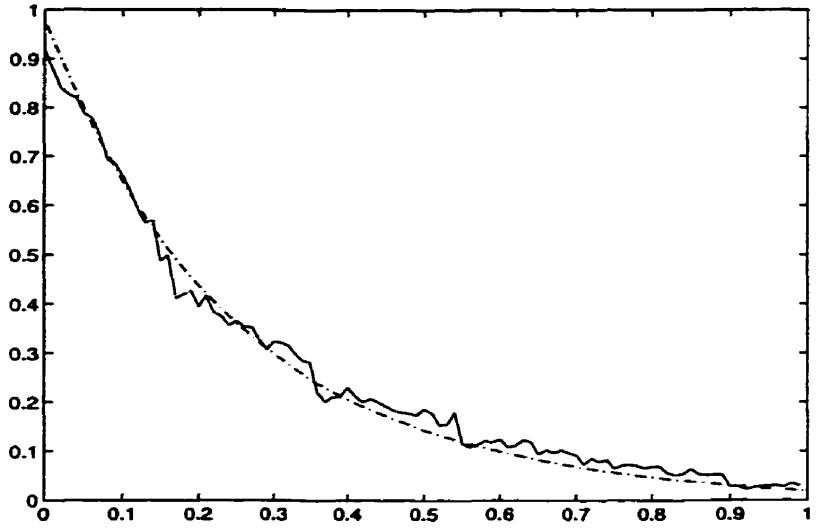
Approximations Found by Fuzzy Cellular Automata.

Figure A.1: Test function: $y(x) = 1.5x^2 - 1.35x + 0.75$.



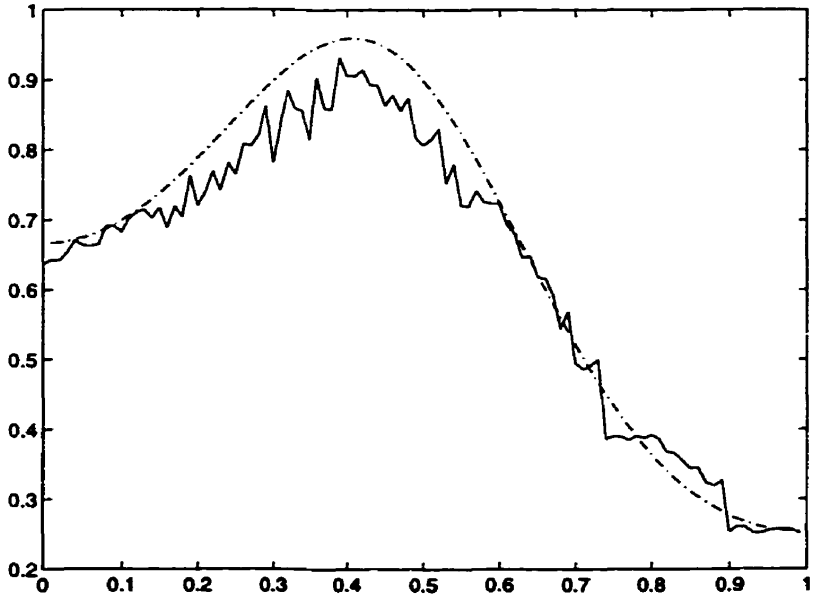
Target function: ".-.-.-"; approximation: "—"

Figure A.2: Test function: $y(x) = \frac{1}{150}\exp(0.5x^2)\exp(0.8(x-2)(x-3))$.



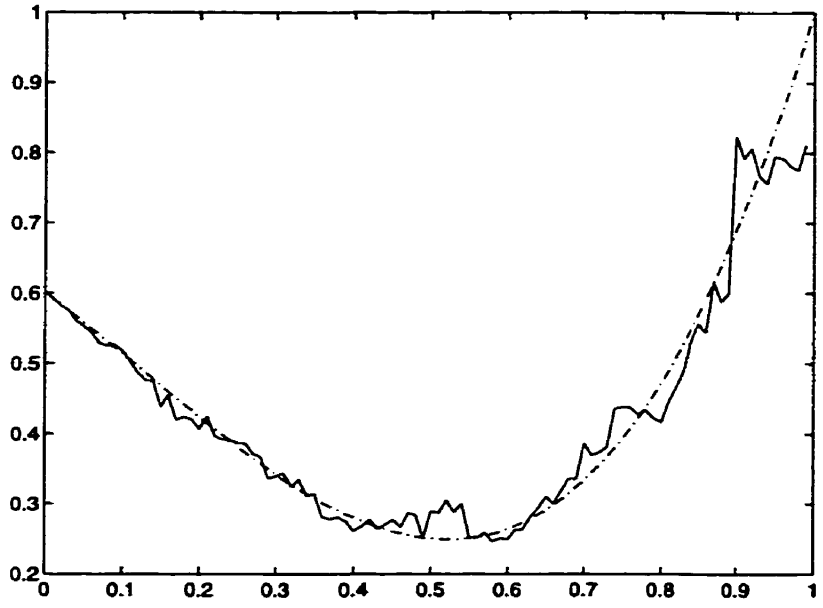
Target function: ".-.-.-"; approximation: "—"

Figure A.3: Test function: $y(x) = \frac{2}{3}\exp(x\sin(5x))$.



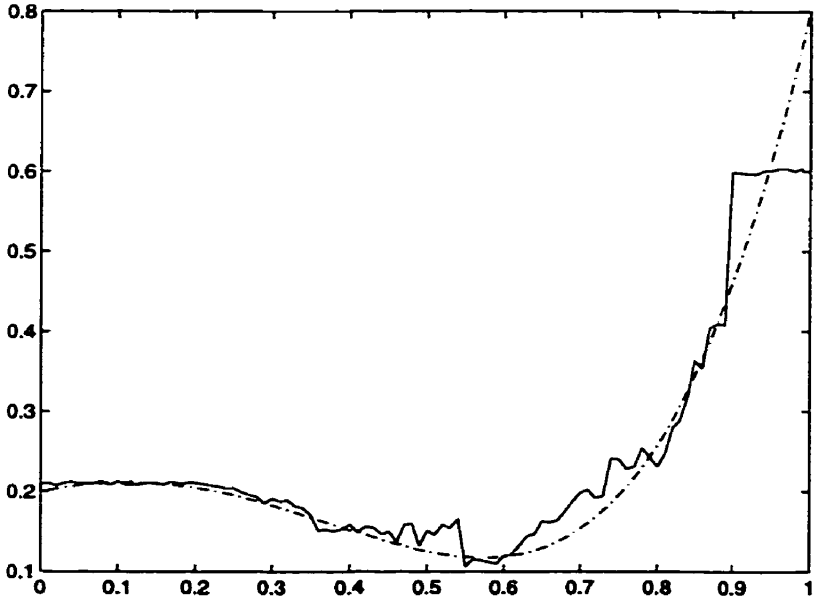
Target function: ".-.-.-"; approximation: "—"

Figure A.4: Test function: $y(x) = 2x^3 - 0.8x^2 - 0.8x + 0.6$.



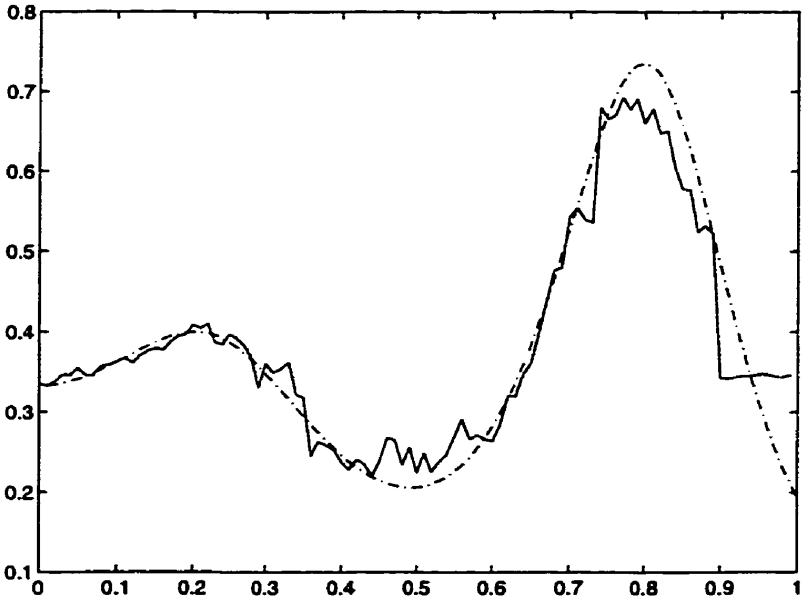
Target function: ".-.-.-"; approximation: "—"

Figure A.5: Test function: $y(x) = 2x^4 - 0.8x^3 - 0.8x^2 + 0.2x + 0.2$.



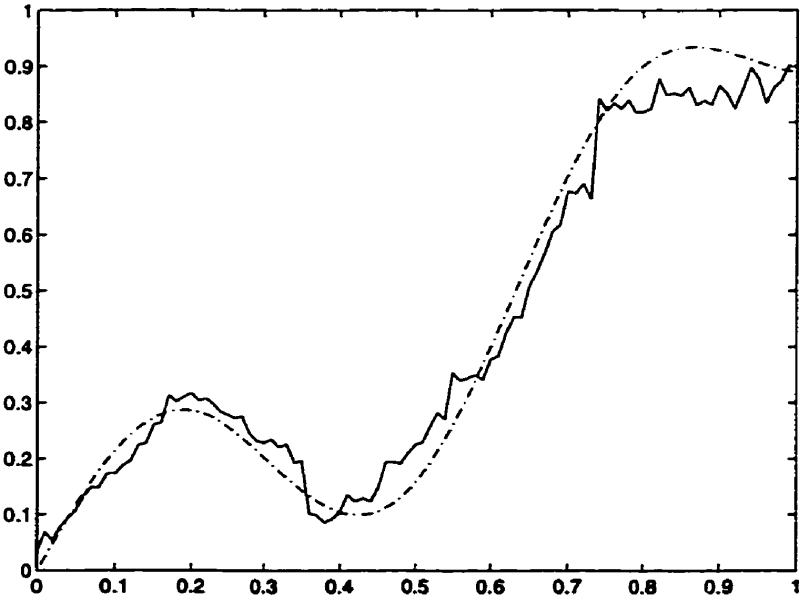
Target function: ".-.-.-"; approximation: "—"

Figure A.6: Test function: $y(x) = \frac{1}{3}\exp(x\sin(10x))$.



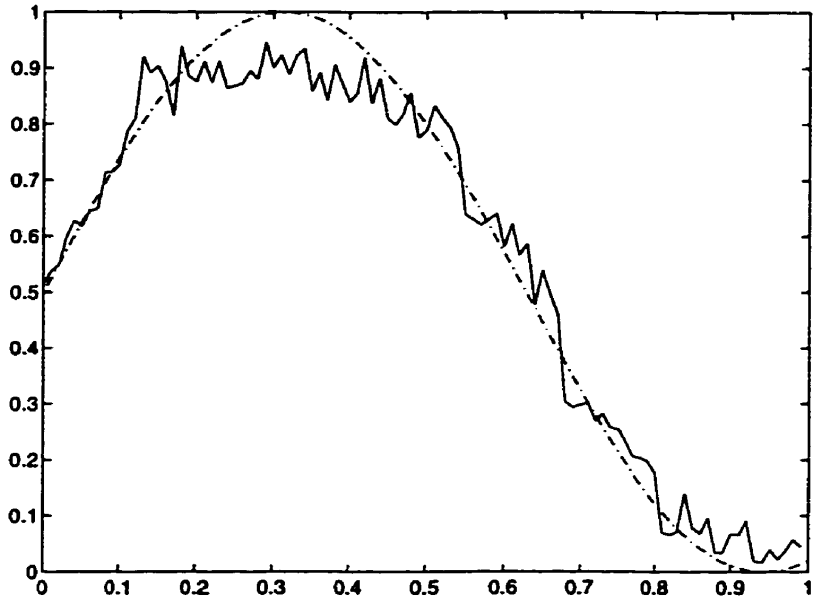
Target function: ".-.-.-"; approximation: "—"

Figure A.7: Test function: $y(x) = \frac{1}{5}\sin(10x) + 3x^2 + 2x$.



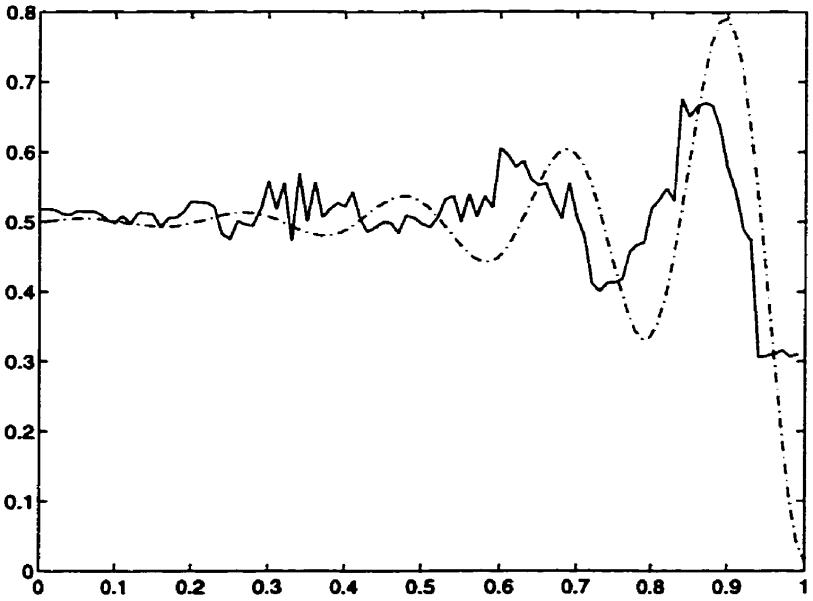
Target function: ".-.-.-"; approximation: "—"

Figure A.8: Test function: $y(x) = \frac{1}{2}\sin(5x) + 0.5$.



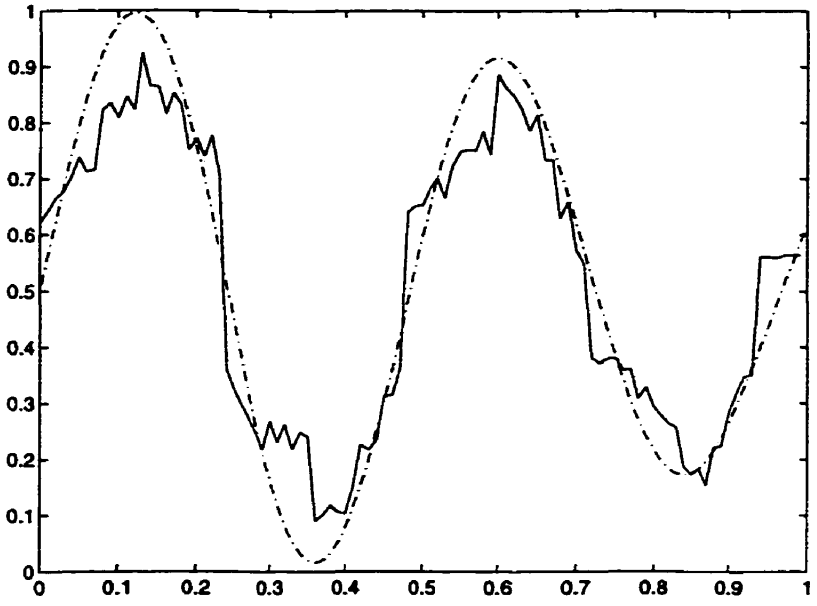
Target function: ".-.-.-"; approximation: "—"

Figure A.9: Test function: $y(x) = \frac{1}{300}(\sin(30x)\exp(5x) + \sin(x)) + 0.5$.



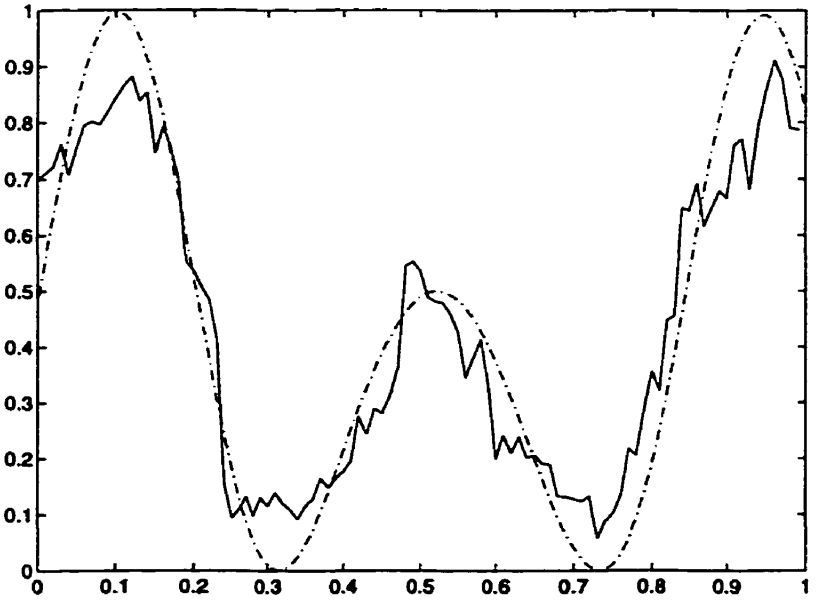
Target function: ".-.-.-"; approximation: "—"

Figure A.10: Test function: $y(x) = \frac{1}{8}\sin(13x)(3 + sin(3x + 1))$.



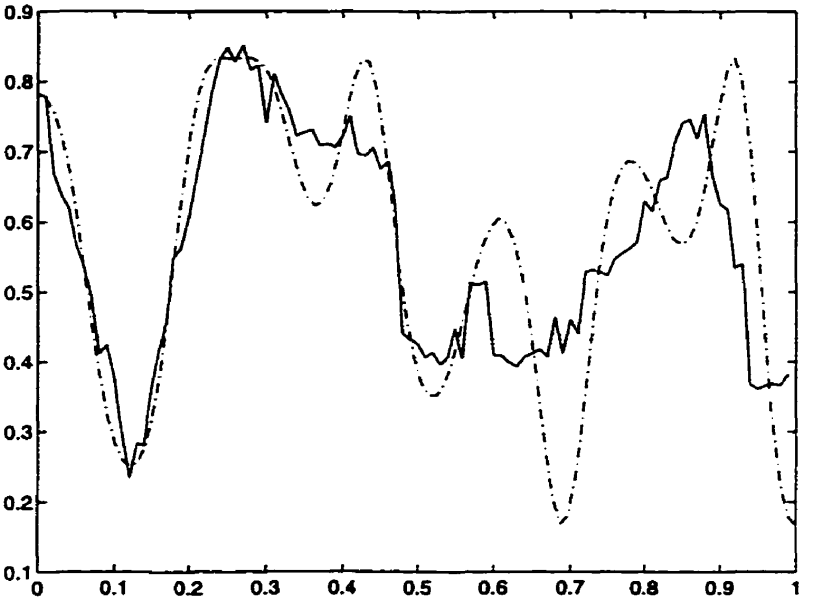
Target function: ".-.-.-"; approximation: "—"

Figure A.11: Test function: $y(x) = \frac{1}{8}(1 + \sin(15x))(3 + sin(7x + 1))$.



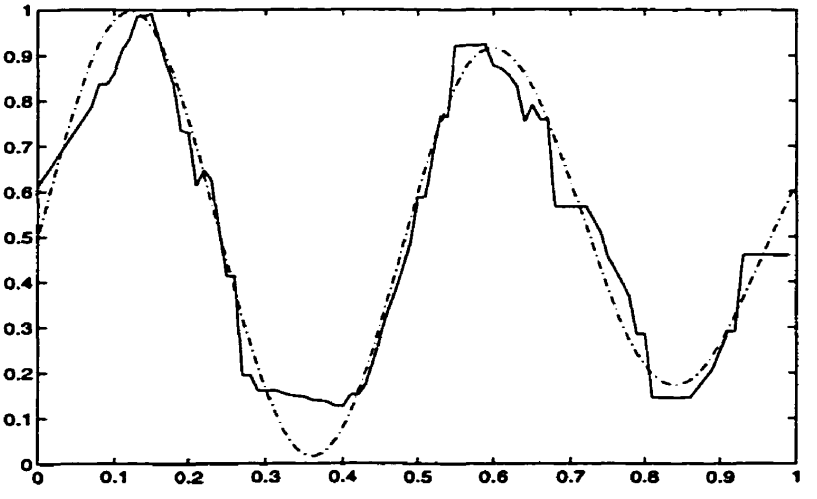
Target function: ".-.-.-"; approximation: "—"

Figure A.12: Test function: $y(x) = \frac{1}{3}\sin(10x^2 + \cos(25x)) + 0.5$.



Target function: ".-.-.-"; approximation: "—"

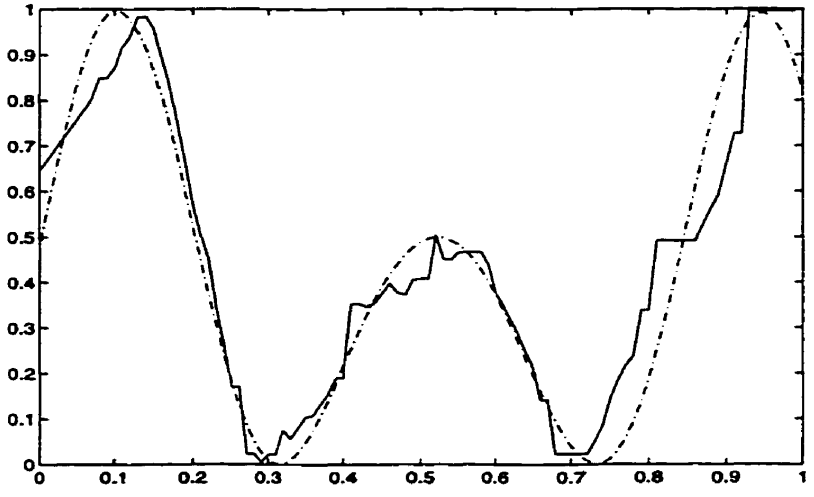
Figure A.13: Test function: $y(x) = \frac{1}{8}\sin(13x)(3 + sin(3x + 1))$.



Target function: ".-.-.-"; approximation: "—".

Cells assume values $\{0, 1\}$ only in an initial configuratopn, encoding a real value.

Figure A.14: Test function: $y(x) = \frac{1}{8}(1 + \sin(15x))(3 + sin(7x + 1))$.



Target function: ".-.-.-"; approximation: "—".

Cells assume values $\{0, 1\}$ only in an initial configuratopn, encoding a real value.

Bibliography

[1] Adamatzky, A., "Identification of Cellular Automata.", Bristol, USA, 1994.

[2] Adamatzky, A., "Identification of Fuzzy Cellular Automata.", Avtomatika i Vychislitelnaja Tehnika, 1991, No.6, pp. 75-80 (in Russian).

[3] Albert, J. and Culik, I., K., "A Simple Universal Cellular Automaton and its One-way and Totalistic Version", Complex Systems 1, 1987, p. 1.

[4] Albin, P.S., "Microeconomic Foundations of Cyclical Irregularities or 'Chaos'", Mathematical Social Sciences, 1987, Vol 13(3), June, pp 185-214.

[5] Banks, E.R., "Universality in Cellular Automata.", In IEEE 11th Annual Symposium on Switching and Automata Theory, 1970, Santa Monica, California, pp 194-215.

[6] Battiti, R., "First- and Second Order Methods for Learning: Between Steepest Descent and Newton's Method.", Neural Computation, 1992, Vol. 4, pp. 141-166.

[7] Batty, M., "Cellular Automata and Urban Form: a Primer.", Journal of the American Planning Association, Spring 1997, Vol.63, pp 266-173.

[8] Bays, C., "Classification of Semitotalistic Cellular Automata in Three Dimensions.", Complex Systems, 1988, 2, p. 235.

[9] Berlekamp, E., Conway, J.H., and Guy, R., "Winning Ways for Your Mathematical Plays.", 1982, Academic Press, New York.

[10] Black, M., "Vaugeness: an Exercise in Logical Analysis.", Philosophy of Science, 1937, 4(4), pp.427-455. (Reprinted in Inter. J. of General Systems, 17(2-3), 1990, pp.107-128).

[11] Bolduc, J.-S., Broderick, G. and Thérien, D., "From Stability to Tracking: Robustness of Cellular Automata Based Controllers.", Proceedings of the 1997 IEEE International Conference on Intelligent Processing Systems, Vol. 1, pp 619-624.

[12] Breiman, L., Friedman, J.H., Olshen, R. A. and Stone, C.J., "Classification and Regression Trees.", 1984, Wadsworth, Belmont, CA.

[13] John von Neumann, "Self-Reproducing Automata", edited and completed by Arthur W. Burks, 1966.

[14] "Machine Learning: Paradigms and Methods." edited by Carbonell, J.G., 1990, The MIT Press.

[15] Cattell, K. and Muzio, J. C., "Synthesis of One-Dimensional Linear Hybrid Cellular Automata.", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.15, No3, March 1996, pp 325-335.

[16] Chaplain, M. A. J. and Anderson, A. R. A., "Mathematical Modeling, Simulation and Prediction of Tumor Induced Angiogenesis.", Invasion and Metastasis, 1996, Vol. 16 (4-5), pp 222-234.

[17] Cherkassky, V. and Mulier, F., "Comparison of Adaptive Methods for Function Estimation from Samples.", IEEE Transactions on Neural Networks, Vol. 7, No. 4, July, 1996, pp 969-984.

[18] "From Statistics to Neural Networks. Theory and Pattern Recognition Applications." Edited by Vladimir Cherkassky, Jerome H. Friedman and Harry Wechsler, 1994.

[19] Codd, E.F., "Cellular Automata", 1963, Academic Press, New York.

[20] Culik, I., K., Hurd, L.P. and Yu, S., "Computation Theoretic Aspects of Cellular Automata", Physica D, 1990, Vol. 45, pp 357-378.

[21] Culik, I.K. and Yu, S., "Undecidability of CA Classification Schemes", Complex Systems 2, 1988, p. 177.

[22] Drazin, R. and Rao, H., "Simplicity and Complexity in Cellular Automata Models of Technological Change.", Technology Studies, 1996, Vol. 3 (1), pp 44-49.

[23] Dubois, D. and Prade, H., "Fuzzy Sets and Probability: Misunderstandings, Bridges and Gaps.", Proc. Second IEEE Intern. Conf. on Fuzzy Systems, San Francisco, 1993, pp. 1059-1068.

[24] Engelen, G., White R., Uljee, I. and Drazan, P., "Using Cellular Automata for Integrated Modeling of Socio-Environmental Systems.", Environmental Monitoring and Assessment, Vol. 34, 1995, pp 203-214.

[25] Ermentrout, G.B. and Edelstein-Keshet L., "Cellular Automata Approaches to Biological Modeling.", J. theor. Biol., Vol. 160, 1993, pp 97-133.

[26] Gardner, M., "The Fantastic Combinations of John Conway's New Solitaire Game "life"", Scientific American, 1970, Vol. 223, no. 4, pp 120-123.

[27] Gill, P.E., Murray, W. and Wright, M.H., "Practical Optimization.", 1991, Academic Press, London.

[28] Gilman, R., "Periodic Behavior of Linear Automata." in Dynamical Systems, ed. J.C. Alexander, Springer Lecture Notes in Mathematics 1342, Springer, Berlin, 1988, p. 216.

[29] Gilman, R., "Classes of Linear Automata.", Ergodic Theor. Dynam. Systems 7, 1987, p. 105.

[30] Gutowitz, H.A., "A Hierarchical Classification of Cellular Automata", Physica D, 45, 1990, pp 136-156.

[31] Gutowitz, H.A., Victor, J.D. and Knight B.W., "Local Structure Theory for Cellular Automata.", Physica D. 28, 1987, p 18.

[32] Haykin, S., "Neural Networks. A Comprehensive Foundation.", 1994, Macmillan College Publishing Company.

[33] Hakman, A. Wan, "Modeling Stock Markets by Probabilistic 1-D Cellular Automata.", Intern. J. Computer Math., Vol. 53, 1994, pp 167-176.

[34] Hopcroft, J.E. and Ullman, J.D., "Introduction to Automata Theory Languages and Computation.", 1979, Addison-Wesley, Redwood City, CA.

[35] Ilachinsky, A., "Topological life games I", Preprint, State University of New York, Stony Book, 1986.

[36] Khuri, A.I. and Cornell, J.A., "Response Surfaces. Designs and Analysis.", 1996, New York.

[37] Klir, G. J. and Bo Yuan, "Fuzzy Sets and Fuzzy Logic. Theory and Applications.", 1995, Prentice Hall P T R, New Jersey.

[38] Klir, G.J. and Parviz, B., "Probability-possibility Transformations: a Comparison", Int. J. General Systems, 1992, Vol. 21 pp. 291-310.

[39] Klir, G.J., "A Principle of Uncertainty and Information Invariance.", Int. J. of General Systems, 1990, Vol. 17, No 2-3, pp. 249-275.

[40] Kramer, A.H., and Sangiovanni-Vincentelli, A., "Efficient Parallel Learning Algorithm for Neural Networks.", In Advances in Neural Information Processing Systems 1., (D.S. Touretzky ed.), 1989, pp 40-48, Morgen Kaufmann, San Mateo, CA.

[41] Lee, Y.C. et al, "Adaptive Stochastic Cellular Automata: Theory.", Physica D, Vol. 45, 1990, pp 159-180.

[42] Medvinskii, A.B., Lysochenko, I.V., Tikhonov, D.A., Tsyganov, V.V. and Ivanitskii, G.R., "Aperoidic Structures in Motile Cell Societies: Mathematic Simulation.", Biofozika 42(2), 1997, pp. 439-448 (in Russian).

[43] Mitchell M., Crutchfield, J. P. and Hraber, P. T., "Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments.", Physica D, 1994.

[44] Mitchell, M. and Hraber, P. T., "Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations.", Complex Systems, Vol. 7, 1993, pp 89-130.

[45] Mizumoto, M., Tojoba, T. and Tanaka, T., "Some Considerations on Fuzzy Automata", J. Comput. Systems Sci., Vol. 3, 1969, pp 409-22.

[46] Mraz, M., Zimic, N., Virant, J., "Predicting Wind Driven Wild Land Fire Shape Using Fuzzy Logic in Cellular Automata.", Proceedings of the Joint Conference on Intelligent Systems, 1996, pp 408-412.

[47] John von Neumann, "The Computer and the Brain", 1958.

[48] John von Neumann, "Probabilistic Logic and the Synthesis of Reliable Organisms from Unreliable Components.", in Collected Works 5.329-378, 1952.

[49] John von Neumann, "The General and Logical Theory of Automata", in Collected Works 5.288-328, 1951.

[50] Nowak, A., Zienkowski, L. and Urbaniak, J., "Modeling Changes in Eastern and Central Europe.", Institute Studiow Spolecznych U Warszawski, 1994.

[51] Olej, V., Chmurny, J. and Lehotsky, M., "Fuzzy Automaton Based on Fuzzy Neurons", in NEURONET '90. Proc. Int. Symp. Neural Networks and Neural Computing, Prague, pp 264-6.

[52] Portugali, J. and Benenson, I., "Artificial Planning Experience by Means of Heuristic Cell-Space Model: Simulating International Migration in the Urban Process.", Environment and Planning A, Vol. 27, Oct., 1995, pp 1647-65.

[53] Qian, F. and Hirata, H., "A Parallel Learning Cellular Automata for Combinatorial Optimization Problems.", Proceedings of the IEEE Conference on Evolutionary Computation, 1996, pp 553-558.

[54] Qian, S. et al., "Adaptive Stochastic Cellular Automata: Applications.", Physica D, Vol. 45, 1990, pp 181-188.

[55] Raghavan, R., "Image Recognition, Learning and Control in a Cellular Automata Network.", SPIE Vol. 1469 Applications of Artificial Neural Networks II, 1991, pp 89-101.

[56] Rosenfeld, A., "Parallel Image Processing Using Cellular Arrays.", Computer, Vol. 16, 1983, p.14.

[57] Shafer, G., "A Mathematical Theory of Evidence", 1976, Princeton University Press, Princeton, N.J.

[58] Sherratt, J.A., Eagan, B.T. and Lewis, M.A., "Oscillations and Chaos behind Predator-Prey Invasion: Mathematical Artifact or Ecological Reality?", Philosophical Transactions of the Royal Society of London Biological Sciences, 352(1349), 1997, pp 21-38.

[59] Shine, L.C. and Grondin, R.O., "On Designing Fuzzy Learning Neural-Automata" in Proc. IEEE First Int. Conf. Neural Networks, Vol. 2, pp 299-307.

[60] Siregar, P., Sinteff, J.P., Julen, N. and Lebeux, P., "Spatio-Temporal Reasoning for Multi-Scale Modeling in Cardiology.", Artificial Intelligence in Medicine, 1997, May; 10(1), pp 41-57.

[61] Siregar P., Sinteff J.P., Chanine, M. and Lebeux, P., "A Cellular Automata Model of the Heart and its Coupling with a Qualitative Model.", Computers and Biomedical Research, 1996, Jun. 29(3), pp 222-246.

[62] Sipper, M., "Evolution of Parallel Cellular Machines. The Cellular Programming Approach.", Lecture Notes in Computer Science, 1997.

[63] Sipper, M., "Co-evolving Non-uniform Cellular Automata to Perform Computations.", Physica D, 1996, pp 193-208.

[64] Smith, A.R., "Simple Computation Universal Cellular Spaces.", Journal of ACM, Vol. 18, 1971, pp 339-353.

[65] Smolle, J., Hofmann-Wellenhof, R., Fink-Puches, R. and Auersperg, N., " Assessment of Tumor Cell Cohesion in Vito Using Pattern Interpretation by Cellular Automata.", Analytical and Quantitative Cytology and Histology, 1996 Jun. 18(3), pp 199-204.

[66] Sutner, K., "The Computational Complexity of Cellular Automata", Proceedings of Fundamentals of Computational Theory, Lecture Notes in Computer Science (Springer, Berlin, 1984), p.451.

[67] Toffoli, T., "Cellular Automata as an Alternative to (Rather than an Approximation of) Differential Equations in Modeling Physics.", Physica 10D, 1984, pp.117-127.

[68] Ulam, S. M., "Electronic Computers and Scientific Research.", in "The Age of Electronics", edited by Overhage, C. F. J., 1962, pp 95-108.

[69] Ulam, S. M., "On Some Mathematical Problems Connected with Pattern of Growth of Figures.", in "Mathematical Problems in the Biological Sciences.", Proceedings of Simposia in Applied Mathematics, 1962, Vol. 14, Providence, Rhode Island, pp 215-224.

[70] Ulam, S. M., "A Collection of Mathematical Problems.", 1960, New York.

[71] Vichniac, G. Y., "Simulating Physics with Cellular Automata.", Physica 10D, 1984, pp 96-116.

[72] Wang, L. and Mendel, J.M., "Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-Squares Learning.", IEEE Transactions on Neural Networks, Vol. 3, No. 5, September, 1992, pp 807-814.

[73] Weaver, W., "Science and Complexity.", American Scientist, 36(4), 1948, pp.536-544.

[74] Wechler, W., "The Concept of Fuzzyness in Automata and Language Theory.", 1978, Berlin: Akademie Verlag.

[75] Wee, W. G. and Fu, K.S., "A Formulation of Fuzzy Automata and its Application as Model of Learning Systems", IEEE Trans. Systems Man. Cyber., Vol.5, 1969, pp 215-23.

[76] White, R. and Engelen, G., "Cellular Automata and Fractal Urban Form: a Cellular Modeling Approach to the Evolution of Urban Land-Use Patterns.", Environment and Planning A, Vol 25, Aug. 1993, pp 1175-99.

[77] Wolfram, S., "Universality and Complexity in Cellular Automata", Physica D 10 (1984) p.1.

[78] Zadeh, L. A. , "Fuzzy Sets", Information and Control, Vol. 8(3), 1965, pp. 338-353.