Neural Networks and Cellular Automata Complexity

J. Gorodkin, A. Sørensen and **O. Winther**
CONNECT, The Niels Bohr Institute
Blegdamsvej 17, 2100 Copenhagen Ø
Denmark
(Final, July 1993)
Email: gorodkin, allan, winther @connect.nbi.dk

Abstract. The genotype-phenotype relation for the 256 elementary cellular automata is studied using neural networks. Neural network are trained to learn the mapping from each genotype rule to its corresponding Li-Packard phenotype class. By investigating learning curves and networks pruned with Optimal Brain Damage on all 256 rules, we find that there is a correspondence between the complexity of the phenotype class and the complexity (net size needed and test error) of the net trained on the class. For Li-Packard Class A (null rules) it is possible to extract a simple logical relation from the pruned network. The observation that some rules are harder for the networks to classify leads to an investigation of rule 73 and its conjugate rule 109. Experiments reveal 3-cycles in magnetization in agreement with observations in higher dimensional cellular automata systems.

1 Introduction

Cellular automata (CA) are dynamical systems discrete in space, time, and state variables, and characterized by possession of exclusively local mechanisms of interaction. They constitute good models for the study of non-linear complex systems. Included among their many applications are simulation tools of biological systems and Artificial Life systems [3], the Navier-Stokes equation of hydrodynamics [5], random number generators [20] and speculative models of everything, i.e., the whole universe being modelled in the form of one single cellular automaton [4]. The popularity of cellular automata stems from their simplicity and transparency in definition; being discrete in all respects they are well suited for computer experiments. But in spite of the simplicity in definition, the set of cellular automata (e.g., the set of one-dimensional cellular automata) contains many rules with very complicated behavior.

It is of interest to characterize, within the set of all CA rules, the location of rules with a certain behavior. The set of rules of a certain behavior—for example, chaotic, constitutes a subset in the set of all CA rules. This subset usually presents a rather non-trivial structure. One way to characterize the structure is through neural networks.

The 256 elementary CAs provide an extremely simple example of a CA system, but there are still a number of unsolved problems. Using neural networks, we study the relation between the genotypes that correspond to the same phenotype class. For each phenotype a network learns to divide the 256 CA rules into two categories: the ones that belong to the phenotype, and the ones that do not. This application of neural networks leads to a further investigation of individual rules and phenotype classes using mean-field theory.

Li and Packard [14] investigated the relation between genotypes of a given phenotype through the probabilities that genotypes are connected to one another, where two genotypes are said to be connected if their Hamming distance is equal to 1. When we use neural networks we also take into account Hamming distances, but the distances are not biased at 1: all possible distances are incorporated by the neural network. Therefore, neural networks provide a more detailed picture of the relation between genotypes of the same phenotype.

We begin by introducing the elementary CAs. Section 3 contains mean-field theory and section 4 treats neural networks and the pruning scheme. In section 5 we present results of neural network simulations, including learning curves and pruning. At the end of the section we discuss the simplest phenotype class and cellular automaton rule 73.

2 The Elementary Cellular Automata

The type of CA considered here is one-dimensional, that is, we have a one-dimensional lattice of cells as illustrated in Figure 1.

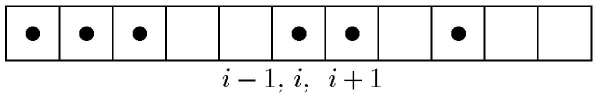


Figure 1: An elementary one-dimensional cellular automata. Each site or cell in the lattice is in either the ON (a filled circle in the cell) or OFF (an empty cell) state. In the example the site $i-1$ is OFF and the sites i and $i+1$ are ON.

Cell states are evolved by updating the ith site, for all i of the lattice, to a new state (ON or OFF), depending on its own state and the two neighbor states at the sites $i-1$ and $i+1$. An example: site i is updated to ON if exactly two of the three sites $i-1$, i and $i+1$ are ON; otherwise it is updated to OFF. We impose a periodic boundary by letting outermost cells be nearest neighbors. The updated string appears in Figure 2.

An update corresponds to one iteration or a single discrete time step. Before any updating procedure is applied it is important to specify the size of the lattice and

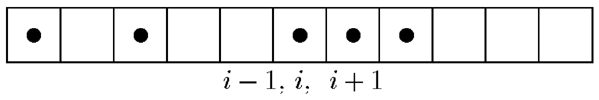


Figure 2: The updated string of cells. Note that all sites of Figure 1 were updated *simultaneously*.

boundary conditions. In theory the lattice can be infinitely long, but in practice it is finite. It is therefore necessary to choose appropriate boundary conditions. They are often periodic, but could just as well be fixed. In general the following must be specified:

- Lattice dimension (here 1D).
- Lattice size (i.e., the number of cells).
- Boundary conditions (i.e., what to do with the edges).
- Initial state of the cells.

For notational purposes we denote ON states by $+1$ and OFF states by -1. In each set of three cells, each cell takes one of two values, -1 or $+1$, and is then mapped to a value which again is either -1 or $+1$. In other words we map $2^3 = 8$ onto 2 possible states. Thus there are $2^8 = 256$ different sets of 3 to 1 mappings. This is illustrated in Figure 3.

-1	-1	-1	$\longrightarrow$	-1
-1	-1	$+1$	$\longrightarrow$	-1
-1	$+1$	-1	$\longrightarrow$	-1
-1	$+1$	$+1$	$\longrightarrow$	$+1$
$+1$	-1	-1	$\longrightarrow$	-1
$+1$	-1	$+1$	$\longrightarrow$	$+1$
$+1$	$+1$	-1	$\longrightarrow$	$+1$
$+1$	$+1$	$+1$	$\longrightarrow$	-1

Figure 3: A procedure which updates a cell state to $+1$ if exactly two of the three cell states are $+1$, and to -1 otherwise. This is just one of the 256 possible updating procedures. The general scheme is to replace the numbers in the right column with numbers $b_j \in \{-1, +1\}$ where $j = 0, \ldots, 7$. From this, all 256 CA rules can be created. (The symbol b_j is chosen for historical reasons.) The table, which contains all the three-blocks is called a *rule table*. Each three block is mapped into a single value by a function $f : \{-1, +1\}^3 \to \{-1, +1\}$ (i.e., $f(-1, -1, -1) = b_0$, $f(-1, -1, +1) = b_1$, ..., $f(+1, +1, +1) = b_7$).

These updating procedures are also called CA rules. The general notation introduced in Figure 3 reads

$$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\ . \tag{1}$$

This notation can be rewritten in Wolfram notation [20] in terms of the b_j as

$$\sum_{j=0}^{7} 2^{j-1}(b_j + 1) \in \{0, \cdots, 255\} \ . \tag{2}$$

From this we see that the CA rule in Figure 3 is CA rule number 104. Any possible 8-sequence of b_j specifies a CA rule. Whether the 8-sequence or the Wolfram CA rule number is used, we call it the *genotype*. Every CA rule regardless of its numbering, is a genotype (i.e., a definition of the local dynamisc applied to the one-dimensional lattice).

The *phenotype* is determined by the asymptotic macrodynamics on the lattice, observed when the genotype has acted for an infinitely long time on a given initial state. (In practice, however we can only observe a finite number of iterations.) Generally, genotypes and phenotypes provide popular terms, and aid intuitive understanding when studying cellular automata.

The phenotypes to be used in the following are the five types suggested by Li and Packard [14]. By numerical simulations on typical initial configurations (i.e., the values of each cell were initially uncorrelated and are taken to be -1 or $+1$ with probability 0.5), five basic qualitative classes of asymptotic global CA behavior were found.

Class A Null rules, 24 elements: Homogeneous fixed-point rules.

Class B Fixed-point rules, 97 elements: Inhomogeneous fixed-point rules.

Class C Periodic rules, 89 elements.

Class D Locally chaotic rules, 10 elements: Chaotic dynamics confined by domain walls.

Class E Global chaotic rules, 36 elements: Rules with random-looking spatial-temporal patterns, or with exponentially divergent cycle lengths as lattice length is increased, or a non-negative spatial response to the perturbations.

An illustration of sample rules from these classes (except the trivial Class A) is given in Figure 4. (The simulations presented in this paper were performed on a HP9000/750 machine with the random generator `drand48`.)

This classification scheme for elementary CAs is similar to the one suggested by Wolfram ([21], for details see [14]). Wolfram [20] "reduced" the number of rules to 88 by considering obvious symmetry operations that never change the phenotypes: reflection, conjugation and the composite operation. We implement these operations on the bit string from (1) in the following.

1. *Reflection* is the interchange of the outermost right and outermost left cells in the three-block of the rule table. This leads to a reflected image in simulations. (e.g., instead of moving to the right one moves to the left), yeilding

$$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0 \ \longrightarrow b_7\ b_3\ b_5\ b_1\ b_6\ b_2\ b_4\ b_0 \ \ . \tag{3}$$

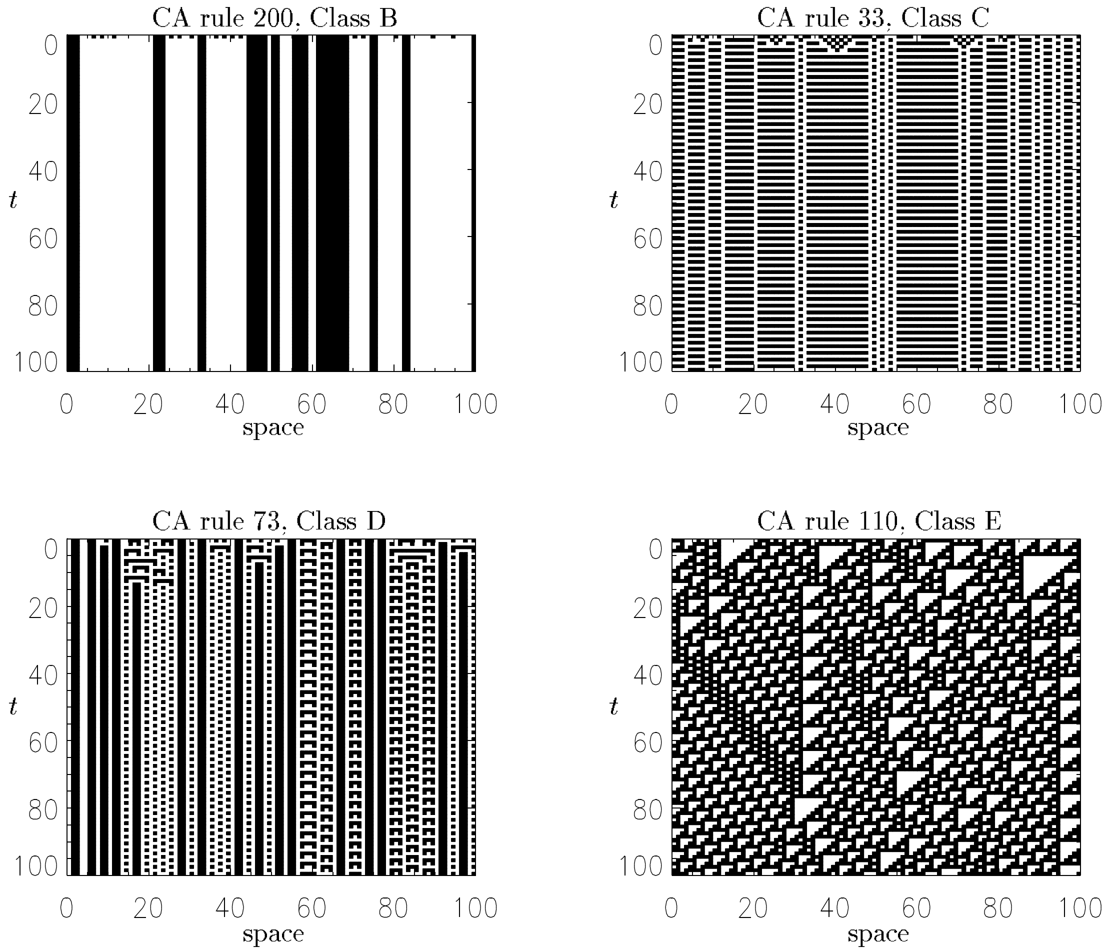


Figure 4: The time evolution of sample CA rules from Class B, C, D and E is illustrated. Note that the time axis is oriented downwards. The horizontal axis represents the one-dimensional lattice. The initial configuration ($t = 0$) were chosen by randomly setting each cell ON with a probability 0.5. (Periodic boundary conditions are chosen.)

2. *Conjugation* is the inversion ($b_j \to -b_j$) of all inputs and all outputs of the rule table, and corresponds to an inverse image in simulation, as follows.

$$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0 \longrightarrow \bar{b}_0\ \bar{b}_1\ \bar{b}_2\ \bar{b}_3\ \bar{b}_4\ \bar{b}_5\ \bar{b}_6\ \bar{b}_7 \quad , \tag{4}$$

where $\bar{b}_j = -b_j$.

3. The commutative operation of combining reflection and conjugation is

$$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0 \longrightarrow \bar{b}_0\ \bar{b}_4\ \bar{b}_2\ \bar{b}_6\ \bar{b}_1\ \bar{b}_5\ \bar{b}_3\ \bar{b}_7 \quad . \tag{5}$$

These operations will later be applied in a discussion of rule extraction from networks.

3 Mean-field theory

In this section we investigate the evolution of a global property of the lattice: the average number of ON states (in other words, *magnetization*). The dynamical state variable at site i, $s_i \in \{-1, +1\}$ evolves according to

$$s_i(t+1) = f\left(s_{i-1}(t), s_i(t), s_{i+1}(t)\right) \quad , \tag{6}$$

where f is expressed in Figure 3. The magnetization at time $t+1$ is expressed as

$$m(t+1) \equiv \frac{1}{N}\sum_{i=1}^{N} \frac{s_i(t+1)+1}{2} = \frac{1}{N}\sum_{i=1}^{N} \frac{f\left(s_{i-1}(t), s_i(t), s_{i+1}(t)\right)+1}{2} \quad . \tag{7}$$

(Note that only ON states are included in the sum.)

The goal is to investigate how well the Li-Packard phenotypes divide among mean-field calculated categories. Inspired by [1] we proceed with the following calculations. The mean-field value in Equation (7) can further be written as

$$\begin{aligned} m(t+1) &= \frac{1}{N}\sum_{i=1}^{N}\sum_{s_1,s_2,s_3=\pm1} \frac{f\left(s_1,s_2,s_3\right)+1}{2}\delta(s_1,s_{i-1}(t))\delta(s_2,s_i(t))\delta(s_3,s_{i+1}(t)) \quad (8)\\ &= \sum_{s_1,s_2,s_3=\pm1} \frac{f\left(s_1,s_2,s_3\right)+1}{2} \times \frac{1}{N}\sum_{i=1}^{N}\delta(s_1,s_{i-1}(t))\delta(s_2,s_i(t))\delta(s_3,s_{i+1}(t)) \quad , \end{aligned}$$

where $\delta(s,s')$ is the Kronecker delta given by

$$\delta(s,s') = \frac{1+ss'}{2} \quad \text{and} \quad \mathrm{s,s'} \in \{-1,1\} \quad . \tag{9}$$

Inserting this value into the previous equation gives.

$$m(t+1) = \sum_{s_1,s_2,s_3=\pm1} \frac{f\left(s_1,s_2,s_3\right)+1}{2} \times \frac{1}{N}\sum_{i=1}^{N} \frac{1+s_1 s_{i-1}(t)}{2}\frac{1+s_2 s_i(t)}{2}\frac{1+s_3 s_{i+1}(t)}{2} . \tag{10}$$

The *mean-field approximation* is then reached by replacing $s_i(t)$ with the mean-value $2m(t) - 1$ (which is 1 for $m(t) = 1$, and -1 for $m(t) = -1$), and thereby neglecting correlation between nearest neighbors. The sum over i vanishes, and

$$\begin{aligned} m(t+1) &= \sum_{s_1,s_2,s_3=\pm 1} \frac{f(s_1,s_2,s_3)+1}{2} \\ &\times \left[s_1 \cdot m(t) + \frac{1-s_1}{2}\right]\left[s_2 \cdot m(t) + \frac{1-s_2}{2}\right]\left[s_3 \cdot m(t) + \frac{1-s_3}{2}\right] \end{aligned} \quad (11)$$

The new sum is over all possible values of the states. Inserting the f values yeilds, after some calculation,

$$m(t+1) = \alpha_0 + \alpha_1 m(t) + \alpha_2 m^2(t) + \alpha_3 m^3(t) \ , \quad (12)$$

where the coeffiecients are given by the matrix equation

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 1 & 0 & 0 \\ 3 & -2 & 1 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \end{bmatrix} \ , \quad (13)$$

with

$$n_0 = \frac{b_0+1}{2} \in \{0,1\} \quad (14)$$

$$n_1 = \frac{b_1+b_2+b_4+3}{2} \in \{0,1,2,3\} \quad (15)$$

$$n_2 = \frac{b_3+b_5+b_6+3}{2} \in \{0,1,2,3\} \quad (16)$$

$$n_3 = \frac{b_7+1}{2} \in \{0,1\} \ . \quad (17)$$

This gives us $2\times4\times4\times2 = 64$ different possible configurations of $[n_0n_1n_2n_3]$. Because the determinant of the matrix in (13) is 1 (and therefore different from zero), each polynomial can thus be represented by a unique configuration of $[n_0n_1n_2n_3]$.

The collection $[n_0n_1n_2n_3]$ is identical to what is usually called the *mean-field* clusters, which is shown clearly in (14)-(17). By investigating the mean-field clusters for each CA rule, we find that Class D behaves differently from the other LP classes in two ways. This is illustrated in Table 1, where those mean-field clusters containing Class-D rules are extracted from the 64 mean-field clusters.

Our first observation is that whenever a rule from Class D appears, it is in a mean-field cluster where all other CA rules are from the periodic Class C. This indicates that rules from Class D could have similar behavior to those from Class C. Interestingly our other observation, illustrated more carefully in Table 2, is that rules 73 and 109 clearly have a different mean-field cluster from the rest of those belonging to Class D. This indicates that these two rules could differ from the rest of the Class with respect to behavior. These observations will be used in a later section.

n_0	n_1	n_2	n_3	Cellular automata rule no. and Li-Packard indices
0	2	1	0	14[C] **26[D]** 28[C] 38[C] 50[C] 52[C] 70[C] **82[D]** 84[C]
1	0	2	0	41[C] **73[D]** 97[C]
1	1	3	0	107[C] **109[D]** 121[C]
0	2	1	1	142[C] **154[D]** 156[C] **166[D]** 178[C] **180[D]** 198[C] **210[D]** 212[C]
1	2	1	1	143[C] 155[C] 157[C] **167[D]** 179[C] **181[D]** 199[C] 211[C] 213[C]

Table 1: The five mean-field clusters of Class D.

CA rule	$[n_0n_1n_2n_3]$
26	[0**21**0]
73	[1020]
82	[0**21**0]
109	[1130]
154	[0**21**1]
166	[0**21**1]
167	[1**21**1]
180	[0**21**1]
181	[1**21**1]
210	[0**21**1]

Table 2: Rule numbers of all the locally chaotic Class-D CAs, together with their mean-field characterization $[n_0n_1n_2n_3]$ (extracted from Table 1). The rules 73 and 109 are the only locally chaotic rules which do not have $[n_1n_2] = [21]$.

4 Neural Networks and Optimal Brain Damage

In this section we describe a specific type of neural network used for determining whether a given CA rule belongs to a particular LP class; together with a method to prune connections called *Optimal Brain Damage* [13].

We consider a network which divides all CA rules into two categories, those which belong to Class A and those which do not. (The same is done for LP class B, C, D and E.) The number of connections is used as a measure of network complexity, even though this measure is not exact.[1] The LP class for which the net with the smallest number of parameters can perform the dichotomy is the one that corresponds to the phenotype for which the relation between the corresponding genotypes is said to have the smallest complexity.

4.1 Application specific networks

An artificial neural network is a network composed of *computational* units (called neurons). The formal neuron is an input/output device which converts a number of inputs into a single output. The neuron has as many *weights* as inputs, and the output F is generally a non-linear *activation* function g of the weighted sum of the

[1]Neural network complexity is an intuitive but still vaguely defined concept. Discussions on using the number of (effective) parameters as a measure of model complexity, for example, can be found in [15, 16] and references therein.

inputs minus a given *threshold*, that is,

$$F = F(b_1, \ldots, b_M) = g\left(\sum_{j=1}^{M} w_j b_j - t\right) \quad ; \qquad (18)$$

where w_j is the weight of input b_j and t the threshold. (Thia neuron is illustrated in Figure 5.) A commonly used activation function is $g = \tanh$ which we will use here; for further details see [12].

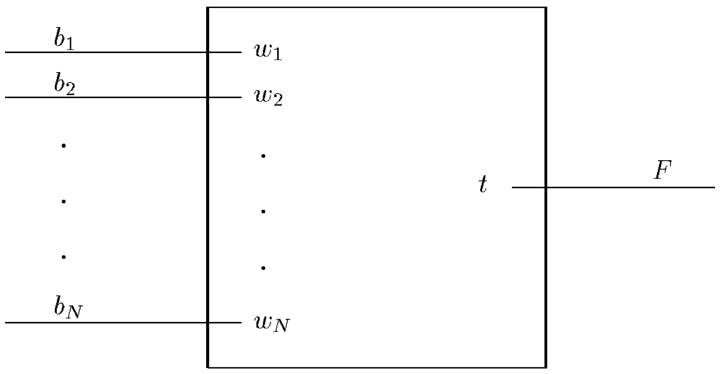


Figure 5: The artificial neuron. The sum of the inputs $b_1, b_2, \ldots, b_M$ is *weighted* each with the weights w_j, $j = 1, 2, \ldots, M$. The threshold is denoted t and the output F. The neuron gets M inputs, which are outputs of other neurons and/or inputs from the outside world.

As the network we use has to answer only "yes" or "no" (i.e., a Boolean problem), we need only one single output; and sa a CA rule can be represented by eight binary numbers, we can be satisfied with eight inputs. Data representation is an important aspect of neural network application, and one could argue that we might as well use 256 inputs and let each input represent a given rule. However, in that case we learn only *an enumeration* of the CA rules and nothing about the specific contents which distinguishes them from each other. In fact using *more* than eight inputs means that we have inserted prior knowledge into the network. We do not want to do that here because we are interested in comparing the nets using exactly the full information from each rule.

After specifying how many inputs and outputs to use, we must determine how to connect them. We use a two-layer feed-forward network as illustrated in Figure 6. It is known that this type of network is capable of universal computation, provided sufficiently many hidden neurons are used [12].

This network implements the function

$$F_w(\vec{b}^\alpha) = \tanh\left[\sum_{i=1}^{M} W_i \cdot \tanh\left(\sum_{j=0}^{7} w_{ij} b_j^\alpha - t_i\right) - T\right] \quad ; \qquad (19)$$

where $\vec{b}^\alpha = b_0^\alpha, \ldots, b_7^\alpha$ is the input, that is, the given CA rule, with $\alpha \in \{0, 1, \ldots, 255\}$. The produced output of the net is $F_w(\vec{b}^\alpha)$. The other quantities are defined in Figure 6. Once the net is trained, in order to get a Boolean output, we apply a hard

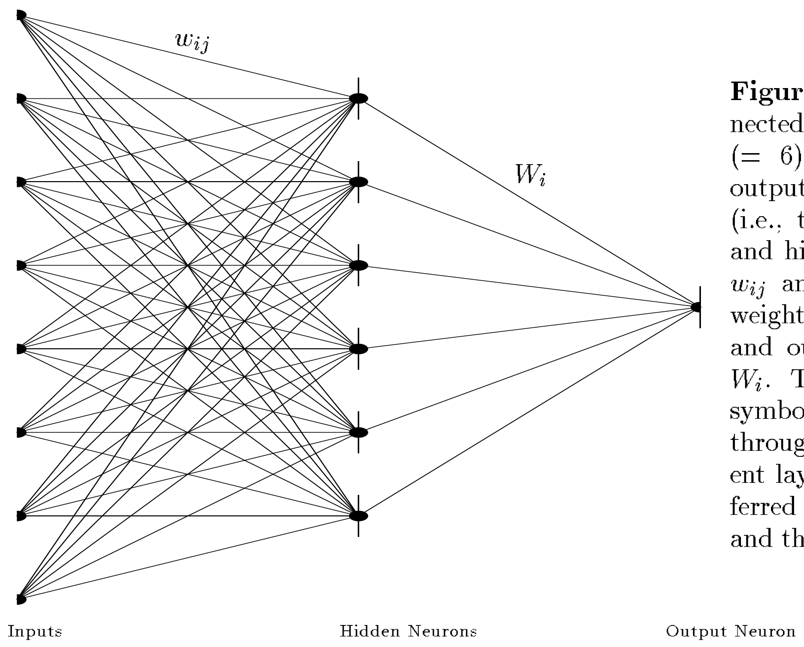


Figure 6: A 2-layer fully connected network with 8 inputs, M (= 6) hidden neurons, and one output neuron. The input weights (i.e., the weights between inputs and hidden neurons) are denoted w_{ij} and the output weights (the weights between hidden neurons and output neuron) are denoted W_i. The thresholds t_i and T are symbolized by the vertical lines through the neurons. The different layers of neurons are often referred to as the input, the hidden, and the output layer.

delimiter on the real-valued output (in this case, a sign-function instead of tanh). So the final function used is $\text{sign}(F_w(\vec{b}^\alpha))$.

As shown in Figure 6, the network architecture is *invariant under a permutation of the input elements*. Therefore, a renumbering of the cellular automata, by permuting b_i and b_j, $i, j = 0, 1, \ldots, 7$, will not change the difficulty of the learning task.

To train the neural net we use the back-propagation learning algorithm (see [12], and references therein), which means that we minimize an error function by gradient descent. It is convenient to choose the error measure as the Kullback relative entropy for tanh output unit [12] given by

$$E(\vec{w}) = \sum_{\alpha}^{\text{tr set}} \left[\frac{1}{2}(1+y^\alpha)\log\frac{1+y^\alpha}{1+F_w(\vec{b}^\alpha)} + \frac{1}{2}(1-y^\alpha)\log\frac{1-y^\alpha}{1-F_w(\vec{b}^\alpha)} \right] \;, \tag{20}$$

where y^α is the known output of input (CA rule) α, and $\vec{w}$ is the vector of all weights (including thresholds). The sum over α includes only those α in the training set.

Breifly, let the vector $\vec{u}$ be the collection of *all* weights in the net. Weight u_k is then updated according to

$$u_k(n+1) = u_k(n) + \Delta u_k(n) \;, \tag{21}$$

where n is the number of iterations (updatings) and

$$\Delta u_k(n) = -\eta \frac{\partial E}{\partial u_k}; \tag{22}$$

in other words, the weights are updated by taking a step in the opposite direction of the gradient scaled with a factor η. This parameter is also known as the *learning rate*. For layered networks the gradient can be rewritten in such a way that it can be back propagated through the network, layerwise.

4.2 Pruning

Careful pruning is most likely to improve the generalization ability of a network that has redundancy in the number of connection (see [13, 9, 6, 19]). Simulations on other tasks have shown that Optimal Brain Damage (OBD) is capable of finding minimal or nearly minimal architectures [6]. The motivation for looking for the minimal representation can be justified by Ockham's Razor [19], a principle that states that

Among two models of the same phenomenon the simplest which decribes it sufficiently well should be preferred.

For a neural network, we will take the number of connections as a measure of the description length, and sufficiently good description will mean giving the right output on the training set. The minimal number of connections needed to perform the mapping tells us something about,in the sense of a neural network, how complex the mapping is.

ODB was introduced by Le Cun *et al.* [13]; the procedure is as follows. First the network is trained to do the training examples, and then the weight with the smallest importance (i.e., the weight which will cause the smallest increase of the error function) is removed. Thereafter the net is retrained and a weight removed again, and so on.

The perturbation of a weight u_k (which is *any* weight in the network) causes a perturbation of the error function (20), which can be written to second order as

$$\delta E = \sum_k \frac{\partial E}{\partial u_k} \delta u_k + \frac{1}{2} \sum_{k,l} \frac{\partial^2 E}{\partial u_k \partial u_l} \delta u_k \delta u_l \ . \tag{23}$$

Because we have learned the training data we are at a minimum with weight $\vec{u}^*$, and the first term can therefore be neglected. The cost of removing the kth weight u_k when trained to $u_k = u_k^*$ (i.e., setting $\delta u_k = -u_k$ and $\delta u_l = 0$ for all $l \neq k$), is then

$$\delta E \approx s_k = \frac{1}{2} \frac{\partial^2 E}{\partial u_k^2} u_k^2 \ , \tag{24}$$

where s_k is called the *saliency* of weight u_k. This is illustrated in Figure 7. How to calculate $(\partial^2 E)/(\partial u_k^2)$, for example, can be found in [13, 6].

5 Numerical Experiments

In this section, we investigate in this section the degree of complexity of the mutual relation between genotypes for each corresponding LP class, using neural networks. We employ two main approaches: comparing learning curves of the respective networks which represent the LP classes, and using the full 256 CA rules as the training set and pruning the nets by OBD. We compare the number of free parameters left in each network.

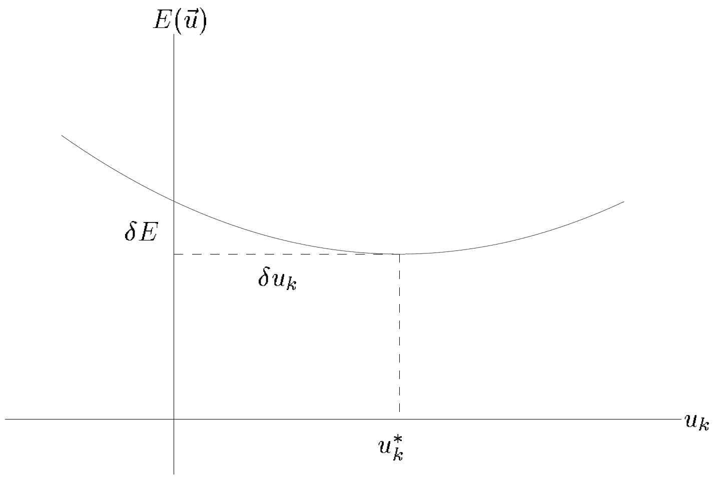


Figure 7: After training, the weight configuration is assumed to give an energy minimum (or close to one) with $u_k = u_k^*$ as illustrated. This weight is removed by setting $\delta u_k = -u_k^*$, that is, $u_k = 0$. The cost of this is δE.

5.1 Learning curves

A learning curve is produced by starting with a given training-set size p_{train}, where the inputs (CA rules) are chosen randomly. The network is then trained until it has learned to correctly classify all examples in the training set. This is done a number of times (which defines the "ensemble size") for different initial weight configurations and increasing training set size p_{train}. We define the test error as

$$E_{\text{test}} = \frac{E_{\text{class}}}{p_{\text{test}} m_{\text{class}}} \; , \tag{25}$$

where the number of misclassifications is

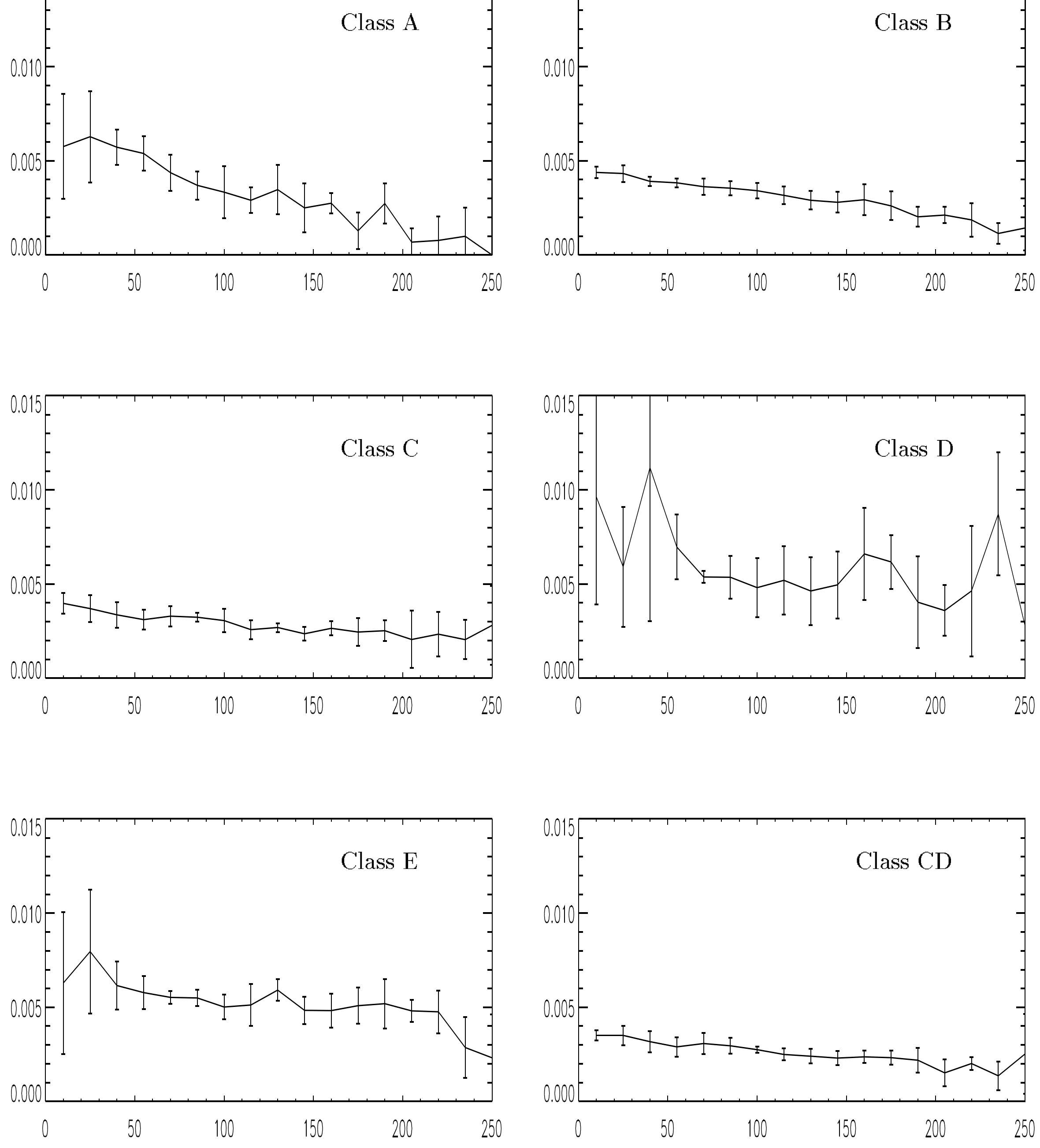
$$E_{\text{class}} = \sum_{\alpha}^{\text{test set}} \Theta\left(-\, y^\alpha F_w(\vec{b}^\alpha)\right) \tag{26}$$

with

$$\Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \; . \tag{27}$$

The misclassifications E_{class} are divided with the test-set size ($p_{\text{test}} = 256 - p_{\text{train}}$), and the number of elements of LP class in question (m_{class}). By dividing over m_{class} we ask how difficult it is to learn the problem per element in the given class. We simply use a *normalization* with respect to the number of elements in each LP class, in agreement with the relative bias of the different networks when respectively trained to recognize a different number of +1s [17].

Letting p_{train} increase, we can plot the average test error of the ensemble for each p_{train} as a function of p_{train}. Such a plot is called a *learning curve*. We use gaussian error bars, even though the errors do not seem to be normally distributed and other more convenient criteria could be used [8, 6]. Figure 8 shows the learning curves corresponding to classification of the different LP classes.



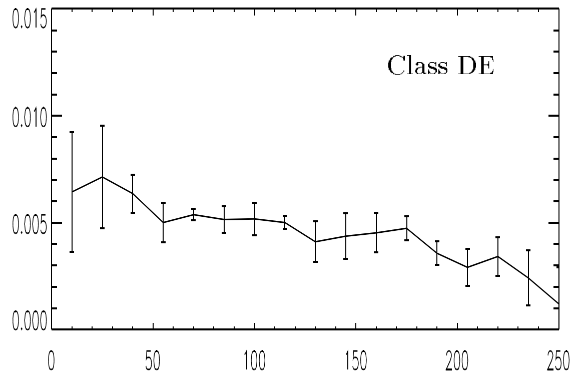


Figure 8: Learning curves for networks representing the CA classes. The x-axis is the number of training examples, and the y-axis is the normalized test error (i.e., the usual test error divided by the number of elements in the respective LP classes). The chaotic classes, indeed, have a much higher error level of their learning curves than the non-chaotic ones. We also see that the behavior of the learning curves for Classes C and CD are similar and, likewise, for Classes E and DE. Each error bar was produced from six nonpruned networks, that is, the ensemble size was 6 networks per training example. Each network had 8 hidden neurons.

The first significant observation from the learning curves in Figure 8 is that the simple LP classe (i.e., A, B, and C) have a much lower level of test error than the chaotic Class D and Class E do. The nets representing the three simple LP classes generalize much better than the chaotic ones do. We see that the relation between genotypes having simple phenotype is simpler than the relation between genotypes having more complex phenotype.

The second significant observation is that Class D merges just as well with Class C as it does with Class E. This is realized by comparing the learning curve of Class C with that of Class CD, and by comparing the learning curve of Class E with that of Class DE. In both cases the two learning curves are "close" to each other. Li and Packard [14] merge Class D and Class E in one case because they are both chaotic and Class D contains few elements. The mean-field theory has encouraged us to merge Class C and Class D.

5.2 Pruned networks

Using the entire set of possible examples, the networks are then pruned as much as possible by OBD, and the resulting number of parameters left in each case are compared. This is reasonable, from the point of view that the number of free parameters in a network can be considered as a measure for how "complex" a function F_{w^*} it implements. Again, we normalize with respect to the number of elements in each LP class. The results are given in Table 3. Each network was found in an ensemble of 15 networks.

Net	m_{class}	q	h	$\frac{q}{m_{\text{class}}}$
A	24	16	3	0.67
B	97	50	6	0.52
C	89	53	7	0.60
D	10	28	3	2.80
E	36	44	6	1.22
CD	99	56	7	0.57
DE	46	48	7	1.04

Table 3: Results for networks corresponding to each of the LP classes A, B, C, D, E and the composites CD and DE. The number of parameters after pruning is q, m_{class} is the number of elements in each LP class, and h is the number of hidden units left after pruning. The initial number of hidden neurons for the networks were for A: 3, for B: 6, for C: 7, for D: 3, for E: 7, for CD: 7 and for DE: 7.

Considering Table 3, we observe that the interesting quantity q/m_{class} is much higher for the chaotic classes D and E than for the non-chaotic classes A, B, and C. Taking into account the number of elements in each LP class, the corresponding networks are thus more complex for the complex LP classes than for the other simpler LP classes. The number q/m_{class} for Class D is very high, which is probably a result of the low number of elements in this class.

A network with the smallest number of parameters does not necessarily have the smallest number of hidden neurons [6]; for Class A we also found a network with 2 hidden neurons and 17 parameters containing interesting symmetries.

The mean-field theory indicated that Classes C and D could merge. As for the learning curves, we see that Class CD merges with Class C at least as well as Class DE merges with Class E. This could suggest that Class D is as much periodic as it is chaotic. On the other hand, it is natural in this regard to investigate how well Class D merges with Class A and B, in order to test whether the networks completely ignore merging of Class D with any other class (i.e., preserve the relative values of q/m_{class}).

For Classes A and D merged, x we found a smallest network with 41 parameters which could solve the classification task. The fraction $q/m_{\text{class}} = 1.21$ together with an $h = 5$ indicate that the two classes cannot be merged; that is, the relations between the genotypes for the different phenotypes do not at all possess the same complexity. For Classes B and D merged, we found a smallest network with 61 parameters (and $h = 7$), hence $q/m_{\text{class}} = 0.57$. This indicates that Class D also merges with Class B; this will be further investigated subsequently.

5.3 A logical relation between Class A genotypes

It is possible to extract a logical rule from the Class A network because the network is saturated (i.e., sign can be substituted for tanh without changing the output of any input). The network which solves the classification task for Class A with two hidden neurons has a remarkably simple structure, as illustrated in Figure 9.

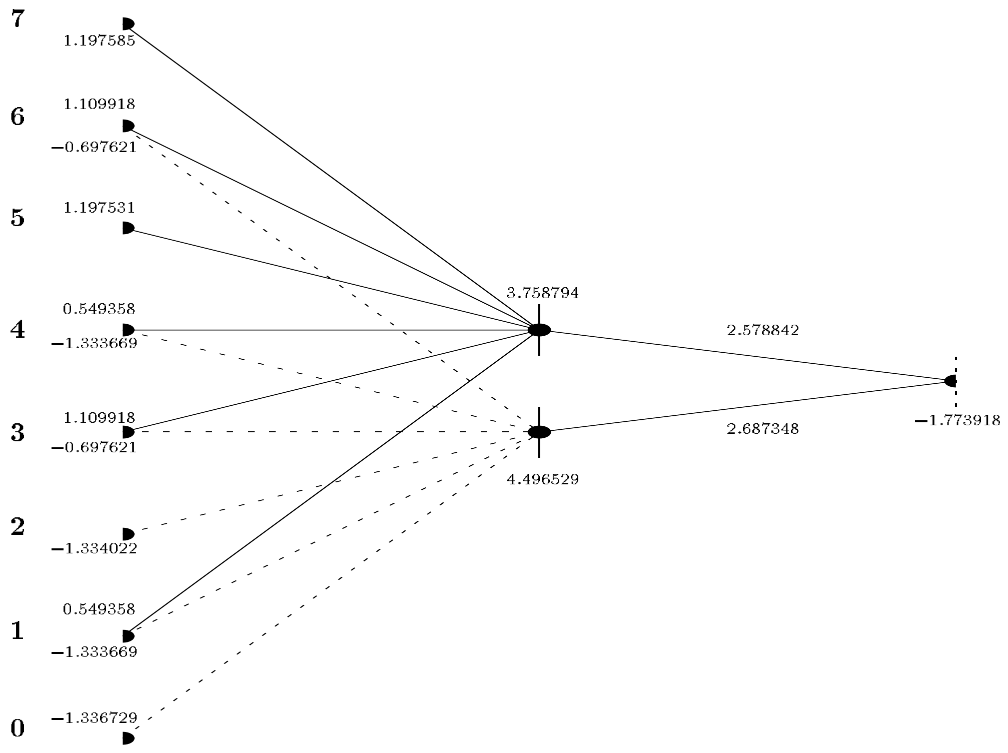


Figure 9: The neural network which can tell whether or not a CA rule belongs to Class A. The dashed lines symbolize negative weights/thresholds. The vertical lines through the neurons symbolize the thresholds. The integers are labels for the inputs and all other numbers are the sizes of the weights. The network is notably *symmetric*.

This network has many identical or nearly identical weights. This *sharing of weights* encouraged a further investigation. The weights which were not identical were set equal and it was established by tests that this new weight configuration (see Figure 10) could produce the right outputs. In the following we will argue why there must be this weight sharing.

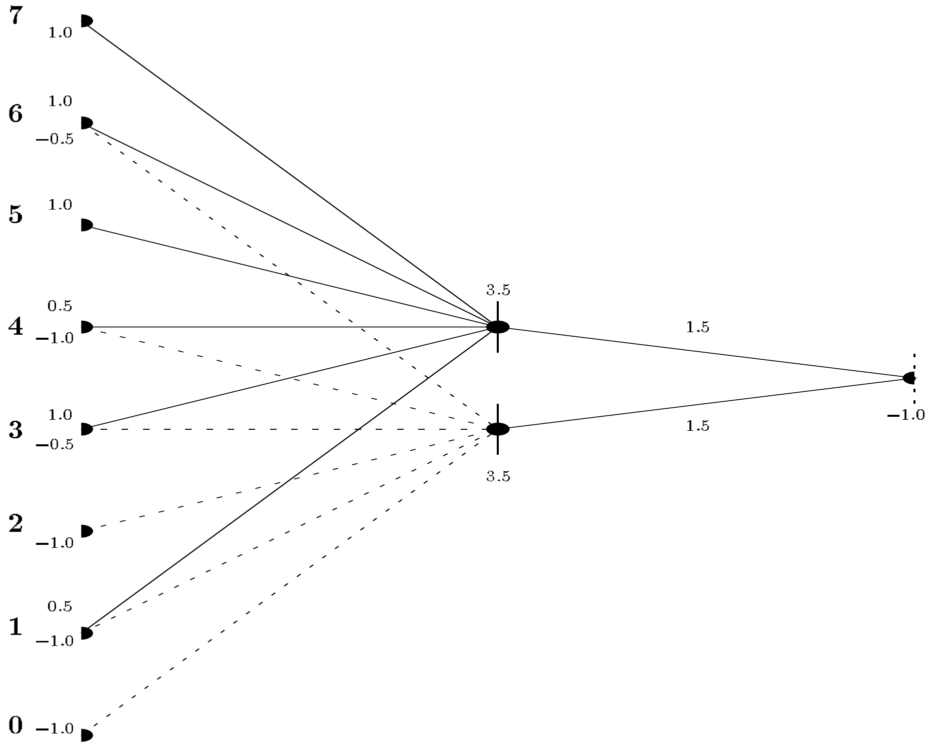


Figure 10: The rewritten network which can solve the classification of CA rules in Class A. There are only four different values for the parameters.

Because the network implements the phenotype classification , it must be invariant under the phenotype conserving symmetries. The network function with two hidden units can be written as:

$$F_w(\vec{b}^\alpha) = g(W_1 g(\vec{b}^\alpha \cdot \vec{w}_1 - t_1) + W_2 g(\vec{b}^\alpha \cdot \vec{w}_2 - t_2) - T) \ , \tag{28}$$

where $g = \tanh$ and $\vec{w}_i$, $i = 1, 2$ are the weights for the respective hidden neurons. Let us define S as a symmetry operation. In the present case,

$$F_w(S\vec{b}^\alpha) = F_w(\vec{b}^\alpha) \ . \tag{29}$$

Both the reflection (3) and the conjugation (4) symmetry have the property that $S^2 = 1$. For the net function (28), this gives two possible choices of weight constraint to conserve the symmetry; on the one hand,

$$S_1\vec{w}_1 = \vec{w}_1 \quad \text{and} \quad S_1\vec{w}_2 = \vec{w}_2 \ , \tag{30}$$

and on the other,

$$S_2\vec{w}_1 = \vec{w}_2 \quad \text{while using} \quad t_1 = t_2 \quad \text{and} \quad W_1 = W_2 \ , \tag{31}$$

where it is easy to see that the latter symmetry operation implies $S_2\vec{w}_2 = \vec{w}_1$. By inspection, we observe that the network has on its own chosen S_1 as the reflection symmetry and S_2 as the conjugation symmetry. It is easy to demonstrate as well that the network is invariant under the combined operation S_1S_2 if (30) and (31) hold. The operation S_2 is completely in agreement with the fact that Class A can be divided into two disjoint sets such that conjugation of all CA rules in one set gives the rules of the other.

Employing the net above, one can extract the following algebraic description, which is implemented by the network.

$$F(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) = \begin{cases} 1 & \text{if } b_0 = b_1 = b_2 = b_4 = -1 \text{ and } b_3 + b_6 \neq 2 \\ 1 & \text{if } b_3 = b_5 = b_6 = b_7 = 1 \text{ and } b_1 + b_4 \neq -2 \\ -1 & \text{otherwise} \end{cases} ,(32)$$

where we note that $b_0 = b_1 = b_2 = b_4 = -1$ and $b_3 + b_6 \neq 2$ gives $n_0 = n_1 = 0$ and $n_2 \leq 2$, and that $b_3 = b_5 = b_6 = b_7 = 1$ and $b_1 + b_4 \neq -2$ gives $n_1 \geq 1$, $n_2 = 3$ and $n_3 = 1$. We recognize that the first two conditions in (32) are each other's conjugates, and both are invariant under reflection.[2] Of course the expression can be written as a logical relation, but that is more cumbersome.

5.4 Borderline cellular automata

A CA rule which is at the border between two LP classes with respect to behavior in a given context (in this case, neural networks) is called a *borderline*. The borderlines were traced by looking for those rules which the networks had trouble with, that is, the most frequent nonlearned rules during the training/pruning process. We present the most frequent borderlines in Table 4. The borderlines for all ensembles except for Ensemble D and CD appear with the same frequency within each ensemble . Rules 73 and 109 clearly differ from the rest of the borderlines in that ensemble. For Ensemble CD it is rule 105 which differs. (Borderlines are also discussed in [18].)

Observation of the run with Classes A and D merged gave the result that 73 and 109 were the only borderlines. This suggest that 73 and 109 are not connected with the rules in CLass A, and that this is the reason why Classes A and C merge so badly. The composite class, Class AD, inherits the borderlines from Class D itself.

Because the Class BD rules 73 and 109 are slightly more common borderlines than the others, it is likely that the elements in Class D are somehow embedded in Class B, due to the large difference between number of elements in the two classes. This could also be the case for Class C and D merging. Nonetheless, the mean-field clusters indicate that the latter merging could be possible, and the network does not *contradict* this.

Though rule 105 is a clear borderline in Ensemble CD, it is not as distinct as 73 and 109 are in Ensemble D.

[2]It should be mentioned that the expression also can be derived directly by observing Class A elements.

Ensemble	N_{Border}	Most frequent non-learned CArules
A_{11}	128_{31}	0[6A] 32[6A] 168[6A] 8[5A] 40[5A] 253[5A] 255[5A] 232[7B] 233[6B] 236[6B] 237[6B]
B_4	72_{47}	128[3A] 234[2A] 254[2A] 255[2A] 139[4B] 57[3B] 69[3B] 116[3B] 209[3B] 201[2C] 60[1E] 106[1E] 153[1E] 195[1E]
C_8	43_{34}	205[2B] 237[2B] 108[3C] 37[2C] 201[2C] 105[4E]
D_5	45_{15}	162[1B] 176[1B] 38[1C] 52[1C] 201[1C] 73[9D] 109[9D]
E_{15}	52_{40}	160[1A] 13[3B] 162[2B] 133[2C] 178[2C] 167[2D] 105[3E] 54[2E] 89[2E] 153[2E] 183[2E]
CD_{14}	47_{26}	4[2B] 72[2B] 132[2B] 197[2B] 113[3CD] 105[8E] 182[3E]
DE_7	34_{26}	94[3CD] 118[2CD] 109[2CD]54[4E] 161[2E]

Table 4: The borderlines for the ensembles from Table 3. For each pruning step of the networks a number of CArules—the borderlines—were no longer learned. The table shows borderlines of back-propagation after 5000 epochs with OBD. "Ensemble" is the LPclass, where the index refers to the number of networks (among the 15 possible) which learned correct classification; in other words, only borderlines for pruned networks are included. The number N_{Border} is the total number of borderlines among the ensemble after the 5000 iterations. The index here refers to the number of *different* borderlines. The numbers with square brackets refer to CArules, together with the number of times they occur in a given ensemble. Letters in the square brackets are the LPclasses.

5.5 Borderlines from Class D

We briefly analyse the borderline rules 73 and 109 which are each other's conjugate CA rules (see (4)). The mean-field clusters indicate that the two CA rules deviate from the other rules in class D. The neural networks find these rules much more difficult to fit in than all the other CA rules. These facts demand a further investigation.

We start by investigating why 73 and 109 are much more frequent borderlines than any other CA rule in Class D. By observing the remaining eight CA rules in Class D and combining them appropriately, we find that the following function produces the right output for all the CA rules except 73 and 109.

$$F(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) = \begin{cases} 1 & \text{if } b_0 = b_2 = b_5 = -1,\ b_1 = b_4 = 1 \text{ and } b_3 = -b_6 \\ 1 & \text{if } b_2 = b_5 = b_7 = 1,\ b_3 = b_6 = -1 \text{ and } b_1 = -b_4 \\ -1 & \text{otherwise} \end{cases} . \quad (33)$$

Because 73 and 109 directly contradict all three claims in the first two cases, it must be clear that incorporating them will increase the complexity of the expression considerably. Even without incorporating them, this expression is still more complicated than the one found for Class A in (32). This is the case in spite of the fact that Class A contains more than twice as many elements as Class D.

Equation (33) gives, in agreement with Table 1, that $n_1 = 2$ and $n_2 = 1$. The first is seen from $b_2 = -1$, $b_1 = b_4 = 1$ or from $b_2 = 1$, $b_1 = -b_4$; and the second from $b_5 = -1$, $b_3 = -b_6$ or from $b_5 = 1$, $b_3 = b_6 = -1$.

There are several ways to investigate the evolution of a CA rule. Inspired by the mean-field considerations we choose to consider here the evolution of magnetization

$m(t)$. By doing so, we neglect how the states are distributed on the one-dimensional lattice, and only consider how many of them are ON and how many are OFF. A first approach towards understanding of magnetization provides the mean-field approximation. Doing so, it is easy to see that rule 73, applied in Equation (13), gives the following polynomial,

$$m(t+1) = -3m^3(t) + 5m^2(t) - 3m(t) + 1 \ , \tag{34}$$

which maps the interval $(0,1)$ into itself. This mapping has one fixed point only, solving $m^*(t+1) = m^*(t)$ with the value $m^* = 0.4056$. Because the right side of the equation leads to a negative *Lyapunov exponent* $\lambda \approx -0.8565$, the fixed point is stable (i.e., an attractor).

The behavior of rule 73 is thus trivial in the mean-field theory. We wished to compare this with numerical experiments, so we investigated rule 73 through numerical simulations of the magnetization. We show a *phase space* plot of the magnetization in Figure 11. We see that it is far from a fixed point, but more like a *periodic* 3-*cycle*, with possible hidden periodicity of higher order.

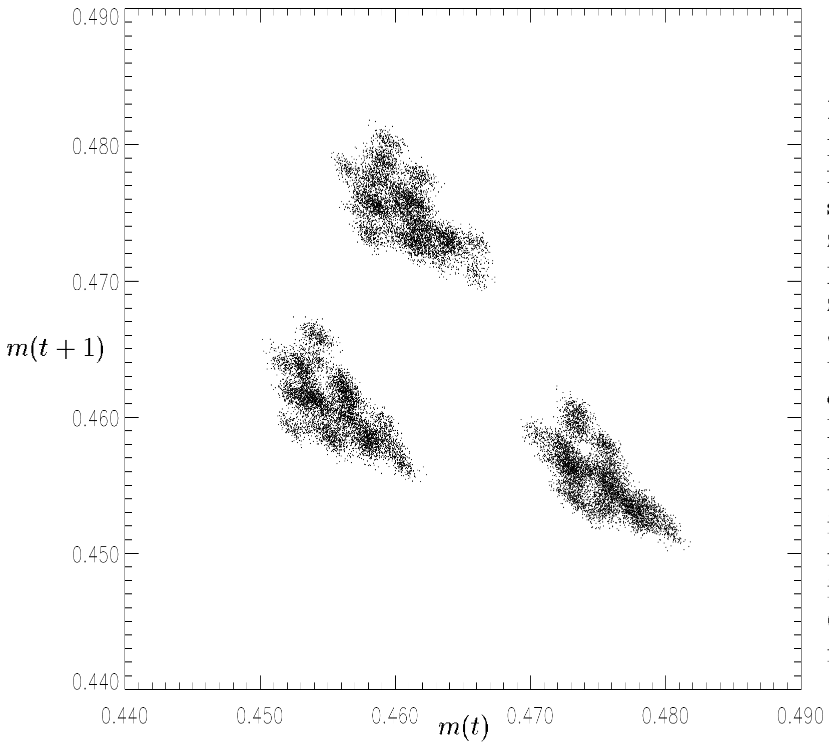


Figure 11: Phase space plot of magnetization for rule 73. We observe how the magnetization at time $t+1$ depends on the magnetization at time t. Of 500000 epochs, the figure shows the last 20,000 data points for a lattice length of 2^{17} with periodic boundary conditions. The initial states in the lattice were set randomly with probability 0.5 for states to be ON, i.e., the initial magnetization was 0.5.

We simulated variable lattice sizes up to 2^{18}, and different random initial cell configurations (with a probability 0.5 for states to be ON) for each. In all cases, the fundamental structure was found to be at the same position; that is, the 3-cycle structure was not just a finite lattice size effect, even though small variations were present for small lattices and a small number of iterations. Furthermore the initial magnetization was crudely varied from 0.2 to 0.8 with steps of 0.1. For initial magnetizations above 0.3 the fundamental structure was preserved, but slightly displaced in the phase space. For initial magnetization 0.2 the 3-cycle structure vanished.

All ten rules of Class D have very similar global patterns on the lattice as considered for instance by Li and Packard. But when magnetization is considered, rules 73 and 109 differ clearly from all the others by having 3-cycles. We also investigated the magnetization for the chaotic Class E rules, and all of them had trivial behavior (though rule 54 displayed an ellipse shaped object that became significantly smaller when the lattice size was increased). The fact that rule 105 also displayed trivial magnetization behavior could indicate that merging Class C and D is not as interesting from a "network" point of view as when seen from the mean-field theory.

Though it is trivial that the periodic rules of Class C display periodic behavior in magnetization, the periodicity in magnetization of the locally chaotic rules 73 and 109 is more subtle. These rules are *a priori* periodic only within the domain walls, but these periodicities are of different length and turned on at different times. The positions of the domain walls themselves are random, because of the random initial configuration of all cells.

The 3-cycle of the one-dimensional rules 73 and 109 seems to be interesting in the ongoing debate about the possibility of producing global oscillations in extended systems with short-range interactions; observations of quasi-periodic behavior in five and three dimensions by Chaté and Manneville [2] and Hemmingsson [10] somehow disagree with the arguments given, for example, by Grinstein [7]. A further discussion of rule 73 in this context is given in [11].

6 Conclusions and perspectives

An important question concerning CAs is the relation between genotypes (rule numbers) having the same phenotype (complexity class). We have studied this relation for the elementary CA rules, using neural networks. Such networks learn by examples, and are known for their ability to generalize and to achieve a compact representation of data.

By applying neural networks in two independent ways, making use of generalization abilities and numbers of connections (net complexity), we have shown that genotypes in the nonchaotic LP classes are connected in a simpler way than the genotypes in the chaotic classes.

Our investigations gave some additional results. We found a logical relation between Class A genotypes, and the networks were able to track down the borderline rules 73 and 109. These most-frequent borderlines revealed a nontrivial 3-cycle in magnetization. Note that not all of the Wolfram Class 4 complex CA rules (i.e., 54, 110, 124, 137, 147, and 193) are capable of universal computation and for that reason, may not be very interesting from a dynamical point of view. This is in agreement with the fact that no special borderline status of these rules was observed.

That the neural networks exposed rules 73 and 109 as borderlines corroborates their differencies from the other rules in Class D with respect to their mean-field clusters. However, searching for intriguing CA rules through mean-field clusters might be cumbersome in higher dimensions; it seems to be much more convenient to use a small number of neural networks.

Whether the results for the one-dimensional elementary CAs hold for higher dimensional systems is an open question. However, there are several directions for future work. 1) Application of the neural network method to classification schemes of cellular automata in higher dimensions. 2) Building symmetries into the networks could *perhaps* lead to logical relations for the other elementary LP classes, and help to minimize the number of free parameters in higher dimensional systems. 3) Construction of new classification schemes in higher dimensions, by neural networks trained with *unsupervised* learning on the space-time evolutions of CAs, in other words, to find clusters in the set of space-time evolutions.

In conclusion, we found that the application of neural networks has led to a nontrivial result relating the complexity of the network learning a specific LP class to the complexity of the dynamical behavior of the LP class itself (chaotic versus nonchaotic). Through the discovery of a metric in the space of CA rules, neural networks are capable of tracking down rules which are on the "edge" of a class. If this holds for higher dimensions, it might be possible to find the universal computational rules at the edge of a chaotic class.

Acknowledgments

We wish to thank Sara A. Solla and Benny Lautrup for valuable discussions, Henrik Flyvbjerg for drawing our attention to mean-field theory, and Lars Kai Hansen for discussions on learning curves; also Allon Percus for proofreading. This research is supported by the Danish Research Councils for the Natural and Technical Sciences through the Danish Computational Neural Network Center (CONNECT).

References

[1] Z. Burda, J. Jurkiewicz and H. Flyvbjerg, *Classification of networks of automata by dynamical mean-field theory.* J. Phys. A: Math. Gen. **23** (1990) 3073-3081.

[2] H. Chaté and P. Manneville, *Evidence of Collective Behaviour in Cellular Automata.* Europhys. Lett. **14** (5), 409-413 (1991).

[3] J. D. Farmer and A. d'A. Belin, *Artificial life: the coming evolution.* Preprint Los Alamos LA-UR-90-378 (1990).

[4] E. Fredkin, *Digital Mechanics: An Information Process Based Reversible Universal Cellular Automaton.* Physica D, Nonlinear phenomena **45** (1990), sept. II.

[5] U. Frisch, B. Hasslacher and Y. Pomeau, *Lattice-gas automata for the Navier-Stokes equation.* Phys. Rev. Lett. **56** (1986) 1505.

[6] J. Gorodkin, L. K. Hansen, A. Krogh, C. Svarer and O. Winther, *A Quantitative Study of Pruning by Optimal Brain Damage*. International Journal of Neural Systems, Vol. 4, No. 2 (June 1993), 159–169.

[7] G. Grinstein, *Stability of Nonstationary States of Classical, Many-Body Dynamical Systems*. Journal of Statistical Physics, **5** (1988), 803.

[8] L. K. Hansen and P. Salamon, *Neural Network Ensembles*. IEEE Transaction on Pattern Analysis and Machine Intelligence, **12**, 993-1001 (1990).

[9] B. Hassibi and D. Stork, *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon*. In Advances in Neural Information Processing Systems 5 (Denver 1992), 164. Editors S. J. Hanson, J. D. Cowan and C. L. Giles. Morgan Kaufmann Publishers, 1993.

[10] J. Hemmingsson, *A Totalistic Three-Dimensional Cellular Automaton with Quasi- periodic Behavior*. Physica A 183 (1992) 255-261, North-Holland.

[11] J. Hemmingsson, A. Sørensen, H. Flyvbjerg and H. Herrmann, *What Synchronization?*. bf 23 (9), pp. 629–634 (1993).

[12] H. Hertz, A. Krogh and R. Palmer, *Introduction to the Theory of Neural Networks*. Lectures Notes Volume I, Santa Fe Institute, Studies in the Sciences of Complexity. 1991 Addison-Wesley.

[13] Y. Le Cun, J. S. Denker and S. Solla, *Optimal Brain Damage*. In Advances in Neural Information Processing Systems 2, 1990, 598-605. Editor D. S. Touretzky Morgan Kaufmann Publishers, 1990.

[14] W. Li and N. H. Packard, *The Structure of the Elementary Cellular Automata Rule Space*. Complex Systems **4** (1990) 281.

[15] J. E. Moody, *The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems*. Appears in J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors. *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[16] J. Rissanen, *Stochastic Complexity and Modeling*. The Annals of Stochastics, 1986, **14**, No. 3 1080-1100.

[17] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley and B. W. Suter, *The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function*. IEEE Transaction on Neural Networks (Letters). **1**, No. 4 1990.

[18] A. Sørensen, *Genotypes and Phenotypes in Elementary Cellular Automata*. Cand. Scient. Thesis, The Niels Bohr Institute, February 1993.

[19] H. H. Thodberg, *Improving Generalization of Neural Networks through Pruning*. International Journal of Neural Systems, **1**, 4. (1991) 317-326.

[20] S. Wolfram, editor, *Theory and Applications of Cellular Automata*. Singapore, World Scientific, 1986.

[21] S. Wolfram, *Universality and Complexity in Cellular Automata*. Physica 10D (1984) 1. 91.