Software Simulation of a Self-Organizing Learning Array System

Janusz Starzyk Zhen Zhu
School of Electrical Engineering and Computer Science
Ohio University
Athens, Ohio 45701, U. S. A.
(740) 593-1580

Abstract:

A neural network paradigm named Self-Organizing Learning Array system has been simulated in software. Hardware implementation limitations are considered in this simulation. Simulation is performed based on a benchmark classification problem, the Australian Credit Card problem. The system behavior is observed and the learning algorithm is examined. The correct classification rate has been compared with some existing classification methods. Although not particularly designed for solving this type of classification problem, this system still shows very good performance. The system will be implemented in FPGA and SOC.

Keywords: Self-Organizing Learning Array, Software Simulation, Self-Organizing Hardware/Fabric

1 Introduction

Complex electronic systems known as Artificial Neural Networks (ANN) have been developed to solve classification and recognition problems. ANN is an interconnected assembly of processing elements (neurons). The processing ability of the network is stored in the inter-connection strengths and transfer functions, obtained by a process of learning from a set of training data. [1]

A neural network should be able to train and organize itself to solve particular problems. Besides each single neuron learning, the network structure also needs to be constructed and updated in the training procedure, for flexibility and generalization. Such a neural network is known as self-organizing system.

Self-organizing system has been an attractive research topic for years. It was mathematically defined in 1960s [2], [3]. The concept of self-organizing has been applied to neural networks. A well-known application is Kohonen's Self-Organizing Map (SOM) [4], which exhibits the self-organizing formation of topographic maps [5]. SOMs are always classified as unsupervised ANN. Besides, other algorithms of Self-Organizing Neural Networks can also be found [6].

A re-configurable neural network design Self Organizing Learning Arrays (SOLAR) has been proposed recently [7], [8]. The hardware structure of SOLAR is similar to programmable arrays such as Field Programmable Gate Arrays (FPGA). The basic frame of SOLAR is a fixed lattice of elemental processing units acting as single neurons, and programmable interconnections between them.

SOLAR array neurons learn from outside inputs and inputs from other neurons processing on the inputs to form their outputs. In addition, neurons are controlled by a threshold-controlled input (TCI), to decide whether they participate in classification at given clock cycle. During training, SOLAR's structure organizes itself and each neuron inside learns basic classification function. Information from each neuron is collected to form the final classification for tested data.

Although the SOLAR network will be mapped to hardware devices to solve complicated real world problems, the whole algorithm has been simulated in software and tested on typical classification problems. The simulation results are prosecuted and analyzed in this paper.

Section 2 demonstrates the details of SOLAR simulation, including neural operations, network structure and classification. SOLAR's performance has been compared with several existing classification methods in Section 3. Section 4 gives brief conclusion suggestions for future work.

2 Simulation of SOLAR

Most of the existing system organizations of neural networks can be defined as feed back (FB) and feed forward (FF) structures. In the FB organization all neurons are generated concurrently. Outputs of each neuron can be connected to inputs of any neuron (including itself). An FB organization may become unstable in case a positive feedback causes the increase in the input signal values. Therefore, a care must be taken to limit a type of operations performed on the input data. We must guarantee that the transformation obtained is contractive, which means that the norm of the output of a neuron must be less or equal to the average norm of all the inputs.

The FF organization is constructed under the assumption that all inputs of each new neuron are connected to the existing neuron outputs or the primary inputs. The neurons are either generated sequentially or in groups. As

new neurons are added, the number of existing neurons increases and the increased number of outputs can be used as inputs in subsequently generated new neurons. In this organization neurons generated later in the process cannot feed neurons generated earlier. An FF organization is always stable.

This paper deals with FF organization of SOLAR system. SOLAR has been simulated with MATLAB. The simulation was carried out with self-organizing hardware design considerations, for instance, hardware computation realization and neural array construction. The simulation was based on a test bench classification problem for artificial systems, Australian Credit Card problem [10]. Simulation results are reported in section 3.

2.1 Neural Input Data

Each neuron in SOLAR system was designed with the ability to perform several operations. One of these operations will be chosen to act on the neuron's inputs. However, the operations need to be carefully defined and handled. The data fed into these operations need to be pre processed.

In general input data is presented to SOLAR with n input features. These n features form the dimensions of the input space. So jth individual input appears as an n-dimension vector: $X^j=[X^j_1, X^j_2 \ldots X^j_n]^T$. Therefore the whole input data set, which consists of s individuals, could be organized in an input matrix,

$$\overline{X} = \{X^1, \ldots X^s\} = \begin{bmatrix} X^1_1 X^2_1 \ldots X^s_1 \\ \ldots. \\ X^1_n . X^2_n \ldots X^s_n \end{bmatrix}.$$

Generally, not all the features are continuous numerical values. Some of them may be in form of discrete symbolic values. If the neuron operations accept only numerical inputs, the symbolic features need to be transformed into real numbers. In practical problems some features of the input data may be unavailable. Default values for each feature needs to be calculated to replace the missing features. This can be done using average values for the missing features selecting values for the missing features which minimize the Mahalanobis distance to the cluster data from a given class, as discussed in 2.1.1. Since all the features are obtained from different measurements, therefore their scale vary greatly. Obviously all the input features have to be rescaled to equalize their significance.

As the result, the pre processing of SOLAR's input matrix will be carried out with 3 steps:
1. Make all the features numerical, set values for symbols of the discrete feature.
2. Determine default values for each features, fill up all the blank items.
3. Rescale all the features to a unified range.

2.1.1 Missing Data Problem

While the average values are easy to calculate, they do not represent well the existing distribution of the training data. This section explains how Mahalanobis distance is used for missing data.

To define Mahalanobis distance we need to use mean value vector for a given class m_c as well as covariance matrix for all training data from this class C_c. This can be easily accomplished using MATLAB simulation. Then a given training data X with missing coordinates can be represented as $X=[X_k, X_m]$ where X_k are known coordinates and the missing values X_m are

$$X_m = \{\tilde{X}_m : d(\tilde{X}_m) = d_{min}\} \text{ where}$$

$$d(X) = (X - m_c)^T C_c^{-1} (X - m_c)$$

Since d(X) is a quadratic form of the unknown values X_m we can find its minimum by setting its derivative to zero

$$\frac{\partial d(X)}{\partial X_m}\Big|_{X_m = \tilde{X}_m} = 0$$

Let us divide the inverse of covariance C_c according to partition of X into known and missing values parts

$$C_c^{-1} = D_c = \begin{bmatrix} D_{kk}, D_{km} \\ D_{mk}, D_{mm} \end{bmatrix}$$

Since C_c is symmetrical $D_{mk} = D^T_{km}$ and

$$\left[\frac{\partial d}{\partial X_1} \ldots \frac{\partial d}{\partial X_n}\right]^T = 2\begin{bmatrix} D_{kk}, D_{km} \\ D_{mk}, D_{mm} \end{bmatrix}\left(\begin{bmatrix} X_k \\ X_m \end{bmatrix} - m_c\right) = 0$$

As a result X_m can be obtained from

$$X_m = -D^{-1}_{mm} D_{mk} (X_k - m_{ck}) + m_{cm}$$

where m_{ck} represents the part in m_c that corresponds to X_k and m_{cm} represents the part that corresponds to X_m.

2.1.2 Pre Processing by Each Neuron

After SOLAR's input matrix has been properly developed, the training data can be sent to the neurons. All the neurons are designed to operate on the full range of input features. Since SOLAR neurons perform local learning, most likely they will inherit data from nearby neighbors. The neurons in the last several layers work on the data that has been processed by many other neurons. In most cases they have their input range poorly condensed. In hardware the shrinking of input range indicates loss of processing precision. Each neuron collects the scaling information during the training procedure and applies it to testing data.

2.2 Simulation of Neural Behavior

SOLAR's neural operations have been designed based on simple elemental operations, for example, identical, exponential or logarithm functions. Combination of the elemental operations could generate complicated expressions, which is expected to separate different classes in the input space and automatically define desired features. However all the operations implemented in

hardware are valid only within certain range of inputs. Each neuron's input has been pre processed, as described in the previous section. This idea is particularly useful if digital computation is used in SOLAR.

Each neuron has pre-defined set of operations which include "unary kernels" like identity, logarithm and exponential, and "binary kernels" like add and subtract operations.

For example, an 8 bit signal has the range 0-255, so its neural operations can be designed as:
Identical function : Y=IDENT(x)= x,

Half function: Y=HALF(x)=$\frac{x}{2}$,

Logarithm function (non-linear):
Y=*NLOG2*(x)=$2^{\log_2(\max(1,\log_2(\max(1,x))))+5}$, as shown in Fig. 1,

Exponential function (non-linear): Y=*NEXP*2(x)= $2^{x/32}$,

Addition function: Y=*NADD*(x1,x2)=$0.5(x1+x2)$,

Subtraction function:
Y=*NSUB*(x1,x2)= $\max(x1-x2,0)$,

Where all function outputs, except for the HALF function, may vary from 0 to 255, just as the input range. Generally the outputs cannot occupy the full range, however. Thus they need to be rescaled. Although in this example digital operations are assumed, they could also be implemented in analog circuits. Based on this set of functions, more complex derivative operations can be generated. A neuron can use either a single unary kernel or two unary kernels combined in a binary kernel.

A neuron may receive inputs directly from SOLAR's input matrix or from previous neurons' output. As indicated in section 1, a neuron is pre wired with redundant data input and threshold control input TCI connections. Using the entropy based metric, known as information index [7], [8], one TCI and one or two input connections are selected with a unary/binary kernel operation for this neuron. A threshold of this operation will be determined to maximize the metric.

Since each operation is a function of data input(s), thus the input space will be separated into two subspaces by using each neuron's threshold value, and the subspace statistical information will be used for final classification decision.

The neuron's input space is different from the SOLAR's original input space, as discussed in section 2.1. It is a transformed part of the original input space. Each neuron transforms the input space using its selected operation. Since neurons in further layers accept the transformed input space from previously generated neurons, their operations on the original input space may become more complicated. An example of subspace division in the original input space is shown in Fig. 2, where SOLAR is simulated based on a 2-D training input database with 5 classes. It can be found that the curve shown in Fig. 2 separates different classes efficiently, but obviously it is not one of the base operations.

In order to find out the relationship between each neuron's input and the original input space, all the neurons record their input connection sources after the training procedure. Each neuron can trace back to find the transformation from the input features to their inputs resulting in an expression of input features X_j and unary/binary kernels.

For instance, a neuron in the second layer operates on its input data as:

$$Operation = NSUB(HALF(Input1), NEXP2(Input2))$$

Input1 and Input2 are defined by neurons in the first layer and substituting for Input1 and Input2 we have:

$$Operation = NSUB((HALF(NADD(NEXP2(X_{14}), NLOG2(X_{11})))), NEXP2(NADD(HALF(X_8), HALF(X_{10}))))$$

or:

$$Operation = \max(0, \frac{1}{4}(NEXP2(X_{14}) + NLOG2(X_{11})) - NEXP2(\frac{(X_8/2)+(X_{10}/2)}{2}))$$

where

$Input1 = NADD(NEXP(X_{14}), NLOG2(X_{11}))$ and

$Input2 = NADD(HALF(X_8), HALF(X_{10}))$ are inherited from the input data connections. Therefore this neuron's local input space based on Input1 and Input2 is a transformed input data space, which is a function of X_8, X_{10}, X_{11}, X_{14}. It has been demonstrated in Fig. 3. This neuron is cutting the local input space with the *operation*. This space is based on a randomly selected set of testing data. Scaling of testing data is based on the information obtained in training. Therefore the data points from a testing set that does not have enough samples may not occupy the whole range of 0-255, as can be found in Fig. 3. Cutting of this input space is performed on a 4 dimensional subspace of the input data space and cannot be demonstrated in a 2-D plot as in Fig. 2.

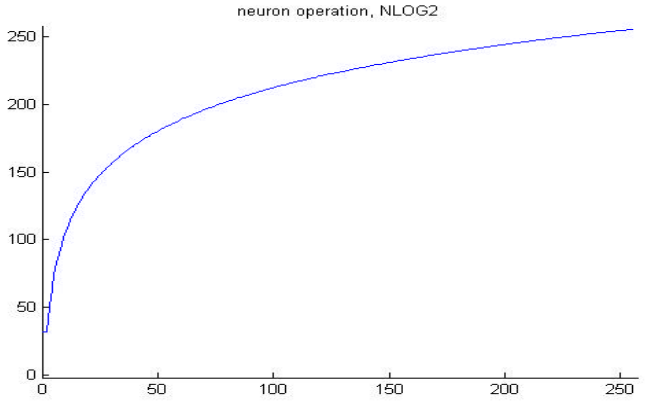

Fig. 1 NLOG2

2.3 SOLAR structure

Each neuron in the lattice is initially pseudo randomly connected to previously generated neurons in the pre-wiring stage. Training procedure will refine this structure. In order to have neurons learn from other neurons' subspaces, it is efficient to connect them with close neighbors in training. Therefore the connection strategy is to connect previously generated close neighbors of each neuron with higher probability. The advantage of this strategy in hardware is that close connections require less wiring spaces and that local features are preserved. Manhattan distance could be used as the metric of neighborhood. In practice a whole layer of neurons in the neuron lattice is generated and trained simultaneously. They only accept data inputs and TCI of neurons from previous layers. Naturally, the first layer neurons are all directly connected to the input nodes.

Self-Organization of the SOLAR system will result in selection of connections between neurons from the initial pre-wired structure. Two or one data input and one TCI connections are reserved for each neuron in the optimized selection. After self-organizing some of the neurons may even be left disconnected, because their input space has little information and additional process cannot contribute to classification any more. This neuron maybe used to solve other classification problems if need. An example of final organization of SOLAR evolved from the pre-wired structure of Fig. 4 is shown on Fig. 5.

2.4 Classification on a Single SOLAR Network

All the neurons inside a SOLAR system work together to classify each testing input in a voting mechanism. After an individual testing data point has been passed through SOLAR, some of the neurons that processed this point will contribute its knowledge with probability based confidence value. The final decision will be made with a weighted voting.

Although a SOLAR classification precision is related to the size of the SOLAR network, it has been found that gradually more and more neurons will not be connected since their information index is low. The size of SOLAR will thus be automatically determined by the learning process and is a function of the number of classes and problem complexity. In hardware this will result in self-determination of the whole network necessary to solve a given problem.

For instance, in the credit card problem, one SOLAR system with 224 neurons provides an averaged correct classification probability of 81.74%. For each testing group, a confusion matrix can $[p_{ij}]$ be obtained:

$$\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1c} \\ \cdots & & & \\ p_{c1} & p_{c2} & \cdots & p_{cc} \end{bmatrix}$$

where c is the number of classes, p_{ij} in the confusion matrix denotes the probability for SOLAR to recognize data of class i as class j while p_{ii} shows SOLAR's correct classification probability for class i.

Although this credit card problem only has two classes to recognize, real world problems can be more complicated.

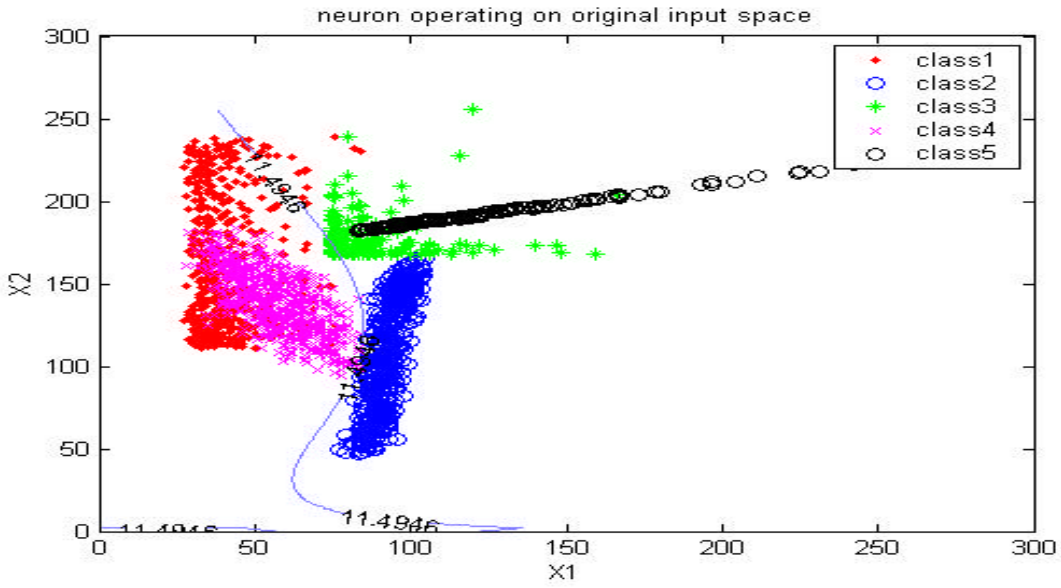

Fig. 2 Neuron Cutting the Original Input Space

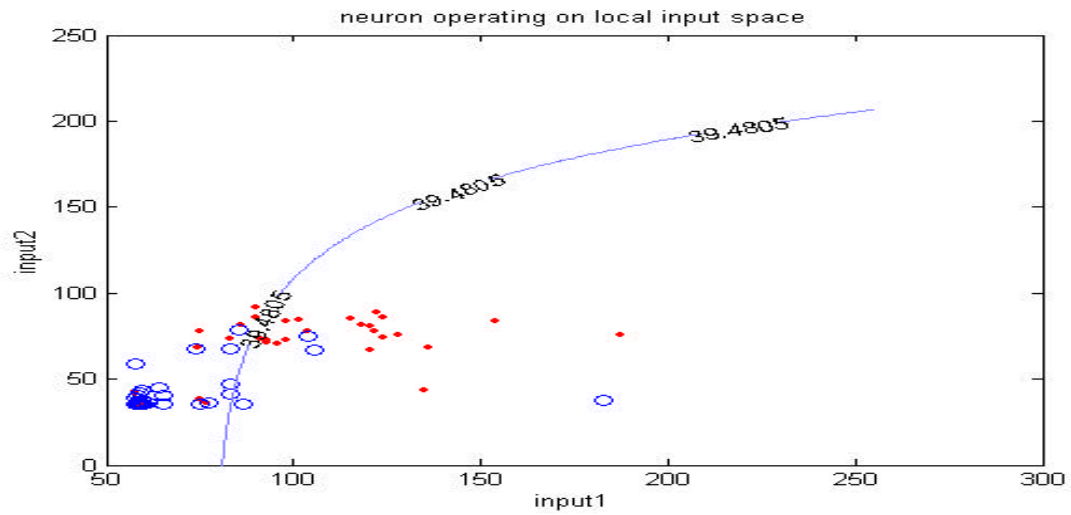

Fig. 3 Neuron Cutting the Local Input Space

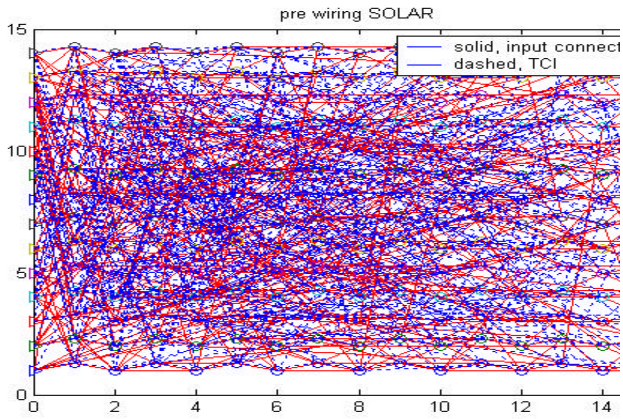

Fig. 4 SOLAR Pre Wiring

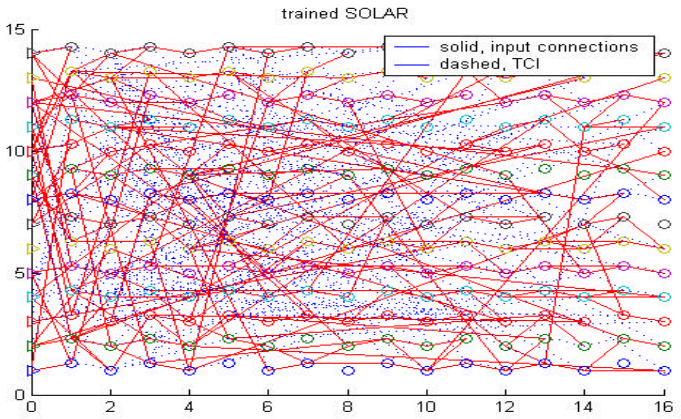

Fig. 5 SOLAR Trained (part and whole picture)

Confusion matrix is a useful measure in a problem with more classes. Some of the classes dominate the training data while other may just have a few training points. Certain trade-offs can be made in SOLAR design. To achieve a high overall correct classification probability, the minor classes may be mostly misclassified as major ones. To maximize the sum of correct classification probabilities for all the classes, significance of training data must be weighted by the inverse of class probability in the training data multiplied by the number of classes.

2.5 SOLAR Ensemble

Compared with other neural network algorithms, SOLAR is easy to construct in hardware and does not require offline training. Classification performance of SOLAR network can be improved in a similar way other NNs.

It has been proven that neural network ensembles are successful in improving the generalization [9]. The ensemble has been realized in a simple version in this work. Several SOLAR networks are trained separately with different parameters in generating lattice. In hardware they will be implemented in parallel.

Since all SOLAR networks are independently pseudo randomly pre wired, their structures are diverse. They work on each testing data point simultaneously and vote for the final classification result. In the straightforward voting scheme, precision has been improved using multi SOLAR ensemble compared with single SOLAR. An example of this ensemble voting has been displayed in Fig. 6.

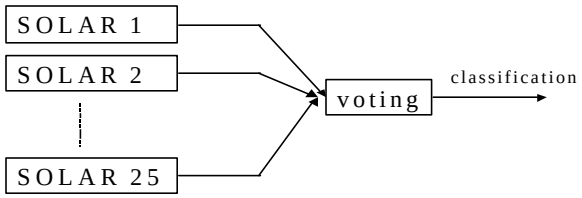

Fig. 6 SOLAR Ensemble

3 Performance Comparison

TO illustrate accuracy of the learning process based on SOLAR concept, a credit card application benchmark [10] was used. This database is available from ftp at cs.uci.edu (128.195.1.1) in directory/pub/machine-learning

databases. The data of 690 individuals were collected and have been sorted into two classes. For performance testing purpose, the whole data set is sub divided into 10 groups, each containing 69 individuals. Each time one group is used to test SOLAR while the others act as training data. The testing procedure has been repeated 10 times, on all the groups [9]. Average correct classification probability signifies SOLAR's precision.

Several traditional classification algorithms have been tested on this benchmark [10], including learning machines, neural networks and statistical methods. Their miss classification rates were reported in the literature and are listed in Table 1 together with results for SOLAR networks.

SOLAR ensemble shows better classification rate than all the listed methods except for CAL5. However, SOLAR acts better than all the neural network methods listed in this table. In addition, decision tree methods, such as CAL5 and C4.5 are believed to have better performance on credit card problems [10] while SOLAR was not specifically design for this case. Other experiments have shown that SOLAR works equally well for arbitrarily assigned training and testing databases.

Method	Miss Detection Probability	Method	Miss Detection Probability
CAL5	0.131	Naivebay	0.151
DIPOL92	0.141	CASTLE	0.148
Logdisc	0.141	ALLOC80	0.201
SMART	0.158	CART	0.145
C4.5	0.155	NewID	0.181
IndCART	0.152	CN2	0.204
Bprop	0.154	LVQ	0.197
Discrim	0.141	Kohenen	-
RBF	0.145	Quadisc	0.207
Baytree	0.171	Default	0.440
ITule	0.137		
AC2	0.181	**SOLAR (single)**	**0.183**
k-NN	0.181	**SOLAR (ensemble)**	**0.135**

Table 1 Miss Rate for Algorithms

4 Conclusion

This computer simulation of SOLAR system tests the algorithm developed to simulate self-organizing learning hardware. SOLAR algorithm has been found to perform well on various classification problems.

It is necessary to observe SOLAR's behavior in experiments before its specific hardware structures are implemented. In this way the algorithm could be examined in details and performance can be estimated. The developed software discussed in this paper provides a testing platform for neural network design. Because of the flexibility of software programming, any types of hardware devices could be simulated with proper models. As the next stage of SOLAR design and simulation, every details of SOLAR organization (for instance, probabilities of neuron connections neuron functionality, signal scaling, threshold control signal selection etc.) will be simulated, and then SOLAR will be implemented initially in Virtex FPGA and finally in a dedicated SOC.

Reference:

[1] Q. He, "Neural Network and Its Application in IR", UIUCLIS—1999/5+IRG, Spring, 1999

[2] Lendaris, G., "On the Definition of Self-Organizing Systems," *Proceedings of IEEE*, v 52, March 1964

[3] Lendaris, G. and G.L. Stanley, "Self-Organization: Meaning and Means" Information Systems Sciences; Proceedings of the Second Congress, Baltimore: Spartan Books, 1965

[4] Kohonen, T., "Self-Organizing Maps" Berlin: Springer-Verlag, 1995

[5] Kohonen, T. "The 'Neural' Phonetic Typewriter", Computer Vol.21 No. 3, pp. 11-22, March 1988

[6] Mulier, F. and Cherkassky, V., "Self-Organization as an iterative kernel smoothing process," Neural Computation, 7, 1165-1177, 1995

[7] J. A. Starzyk and Y. Guo, "Entropy-Based Self-Organized ANN Design Using Virtex FPGA", the International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, NV, June, 2001

[8] J. A. Starzyk and J. Pang, "Evolvable Binary Artificial Neural Network for Data Classification", the 2000 International Conference On Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, June 2000

[9] Y. Liu, X. Yao and T. Higuchi, "Evolutionary Ensembles with Negative Correlation Learning", IEEE Trans. on Evolutionary Computation, Vol. 4, No. 4, Nov 2000.

[10] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine Learning, Neural and Statistical Classification" London, U. K. Ellis Horwood Ltd. 1994