Self-Organizing Learning Array

Janusz A. Starzyk, *Senior Member, IEEE*, Zhen Zhu, and Tsun-Ho Liu

***Abstract*—A new machine learning concept—self-organizing learning array (SOLAR)—is presented. It is a sparsely connected, information theory-based learning machine, with a multilayer structure. It has reconfigurable processing units (neurons) and an evolvable system structure, which makes it an adaptive classification system for a variety of machine learning problems. Its multilayer structure can handle complex problems. Based on the entropy estimation, information theory-based learning is performed locally at each neuron. Neural parameters and connections that correspond to minimum entropy are adaptively set for each neuron. By choosing connections for each neuron, the system sets up its wiring and completes its self-organization. SOLAR classifies input data based on the weighted statistical information from all the neurons. The system classification ability has been simulated and experiments were conducted using test-bench data. Results show a very good performance compared to other classification methods. An important advantage of this structure is its scalability to a large system and ease of hardware implementation on regular arrays of cells.**

***Index Terms*—Information theory-based machine learning, multilayer learning array, self-organizing neurons.**

I. Introduction

ARTIFICIAL neural networks (ANNs) were introduced with the hope to model functions and performance of the human brain. However, a very little progress was made toward implementation of structures and distributed learning algorithms that would yield essential features of neural processing easily observable in a living brain. We need to close the gap between what is understood from experimental neuroscience and what can be done in engineered networks of artificial neurons. Most of the algorithms developed in the field of artificial intelligence do not translate well into concurrent structures. They require either global evaluation criteria (like a measure of fitness function in a support vector machine [1]) or global representation of the machine state (like in defining Markov decision process for reinforcement learning [2]) or both. None of these can be implemented in distributed structures of neurons.

Following the current path of developing machine learning methods for the von Neuman type of machine prevents us from gaining insight into how the brain may work and organize itself. Instead of following this path, we shall focus on organization of networks of neurons and neural interaction, rather than algorithmic (software driven) solutions to machine learning, because processing signals in a network of neurons increases computational speed, improves fault tolerance, constraints us to use distributed solutions, because the brain does it this way.

This requires development of general-purpose learning networks that learn without an algorithm, run without software and are data driven. Such networks shall self-organize, learn through associations, and act with self-awareness. Such networks must scale to very large systems, and be fault tolerant and modular for easy implementation in existing technologies: either dedicated very large scale integrated (VLSI) chips or field programmable gate array (FPGA) structures. Finally, such networks shall learn by interaction with environment, be able to analyze results of their actions, classify these results according to a trained principle and modify their behavior to improve the outcome of their actions. In other words, the machine must be organized to do good, and while learning what is good, drive itself to accomplish the good results. While we are far away from realizing these objectives, this work is aimed to implement some of them.

Several types of machine learning systems have been introduced, including ANNs, cellular neural networks (CNNs), decision trees, genetic algorithms, reinforcement learning, computational learning theory, etc. [3]. Since the approach presented in this paper is most related to ANNs and CNNs, a direct comparison with these networks is provided. In addition, due to the information-based training, self-organizing learning array (SOLAR) is compared to probabilistic neural networks [4].

A neural network learns by adjusting the interconnection strengths and transfer functions, using information contained in the training data. Generally, there are two types of learning algorithms for ANNs: supervised and unsupervised [5]. Supervised neural networks are provided with desired outputs in training, while unsupervised are not. Neural network organizations can be classified into three categories according to different structures, CNNs, feedforward neural networks (FFNNs) and feedback neural networks (FBNN) [6].

The diversity of target problems solved by neural networks requires adaptability and flexibility of their organization. Self-organizing ANNs have been used to satisfy this requirement. These systems are able to construct and update themselves throughout training. The self-organizing system was mathematically defined in the 1960s [7]. The concept of self-organization has been applied to Kohonen's self-organizing map (SOM) [8] classified as unsupervised ANN. Adaptive self-organizing network based on the cerebellar model articulation controller (CMAC) has been used in control applications and other neural network problem domains [9], [10]. For instance, a CMAC network based self-organizing classifier design has recently been reported in [11]. Yet another example of the self-organizing learning networks—self-organizing controller (SOC), introduced by Procyk and Mamdani [12] was used in applications of self-organizing fuzzy control.

Manuscript received March 15, 2003; revised April 23, 2004. This work was supported in part by the School of Electrical Engineering and Computer Science, Ohio University, and in part by a Grant from Xilinx Inc.

The authors are with the School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 USA (e-mail: starzyk@bobcat.ent.ohiou.edu).

Digital Object Identifier 10.1109/TNN.2004.842362

In this paper, a new, supervised SOLAR is proposed. While SOMs self-organize the training data to obtain clusters of self-similar inputs, SOLAR self-organizes its hardware resources (interconnect structure and neural function) to perform classification and recognition tasks. While SOMs use global information (data are organized in the whole input space), SOLAR performs local operations on the transformed subspaces of the input space. Similar to the structure of CNNs, SOLAR has a fixed array of elemental processing units acting as single neurons, and programmable interconnections between them. Initially, SOLAR neurons are randomly connected to other neurons. Controlled by signals from other neurons, they perform basic transformations of their input signals. A neuron's parameters and connections are reconfigured as a result of training, and effectively, SOLARs structure self-organizes establishing its final wiring. The SOLAR structure is aimed at hardware implementation in FPGAs. In this paper, a feedforward version of SOLAR with static neurons is presented with its software simulation. It will be demonstrated that SOLAR yields an efficient hardware realization as well as precise classification.

In what follows, Section II presents SOLAR organization. Behavior of a single neuron is introduced in Section III. Section IV describes neuron self-organization. Output classification voting is discussed in Section V. Section VI shows classification results using several popular test benches and compares SOLAR with other algorithms. Section VII is a brief conclusion.

II. SOLAR Organization

SOLAR is a hardware-oriented, local learning method where neurons learn in parallel to separate data in their individual output spaces using mutual-information objective function based on subspace entropy. This is similar to traditional neural networks, but in contrast to global learning methods like support vector machines [1], clustering [13], or SOMs. Statistical results of multiple neuron learning are pulled together in the final voting using estimated class probabilities. Unlike many popular learning machines, including back propagation ANNs [14], adaptive resonance theory [5], learning vector quantization [5], Kohonen's SOM, or various decision trees [15], SOLAR has a deep multilayer structure, where information about training data is distributed. This structure is more hardware efficient for learning complex relationships between features extracted from training data, than a structure with two or three layers.

Existing hardware implementations of learning machine systems are mostly based on FPGAs [16], [17] and analog VLSI designs [18], [19]. SOLAR can be implemented in both platforms. It should be noted that SOLAR neurons are sparsely connected, which is different from classical ANN models but similar to the sparsely connected Hebbian networks proposed in [20]. In classical ANN models few layers are used, and each processing unit at each layer is either interconnected to all other units in the network or all units at the adjacent layers. Such wiring approach causes the area consumed by interconnections to be far greater than that of the processing units [21]. The regular lattice structure and sparse wiring strategy of SOLAR saves a remarkable amount of wiring area on silicon compared with classical designs without reduction in performance quality.

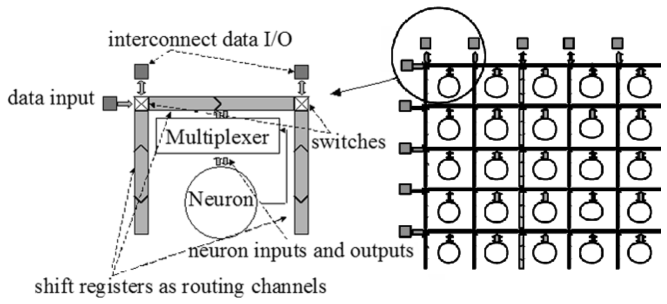

interconnect data I/O
data input
Multiplexer
switches
Neuron
neuron inputs and outputs
shift registers as routing channels

Fig. 1. Basic SOLAR structure.

Since SOLAR bears resemblance to CNNs, let us define it in relation to CNNs, pointing similarities and differences between the two structures.

Definition: SOLAR is a regular, two or three-dimensional (2-D or 3-D) array of identical processing cells, connected to programmable routing channels. Each cell in the array has ability to self-organize by adapting its functionality in response to information contained in its input signals. Cells choose their input signals from the adjacent routing channels and send their output signals to the routing channels.

A SOLAR structure in many ways resembles the organization of a CNN. Like in a CNN, its architecture is defined by an array of identical cells, which adapt their behavior to the input data. Its neurons are cellular automata, which can be programmed to perform different computational tasks based on data received from its neighbors. Neurons can be either static or dynamic, as defined in [22], depending on the type of equation solved. However, unlike in a CNN, its connectivity structure is not fixed. In a CNN, the interconnect structure is defined by templates which limits its learning ability, while in a SOLAR the interconnect structure is an element of learning and can by dynamically changed even during the network's operation. Thus, a CNN can be considered as a special case of SOLAR structure.

The SOLAR implementation presented in this paper employs a feedforward (FF) structure with all neurons arranged in multiple layers. Using the routing channels, neurons are connected to the original inputs or the outputs of neurons from previous layers. The neurons are trained in parallel using their input signals, and send their output signals to other neurons. Feedback SOLAR structures and the structures with dynamic neurons are currently under study.

Based on the dimensionality and complexity of input data, a variable number of layers and neurons are used. This is automatically decided by the learning algorithm. For simplicity of hardware implementation, it is assumed that a fixed number of neurons are added per each layer, thus, an array of neurons is formed. Typically, the number of neurons per each layer is set equal to or greater than input data dimensionality, and the number of layers is automatically determined by the training algorithm. Each neuron is identified by its row and column location, as shown in Fig. 1 and $N_{k,l}$ refers to the neuron in the kth row and lth column (or layer). The regular organization better utilizes available silicon area. All the neurons inside the array are prewired with redundant data and control connections. Training procedure refines the connections and establishes the final SOLAR wiring structure. This organization is well suited

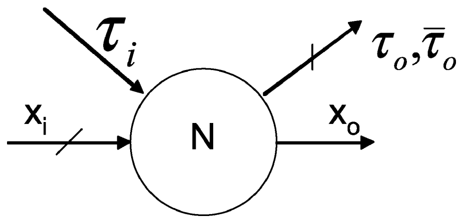


Fig. 2. Neuron input and output signals.

for implementation in FPGA [23] due to its regular cell array structure and local, sparse interconnect.

It is believed that biological neurons tend to have local connections [24]. Therefore, in SOLAR, there is a higher probability that a neuron connects to close neighboring neurons. In the feedforward structure, new neurons are connected only to the previously generated ones. Although local connections are preferred, some more distant connections are randomly used in the prewiring stage with smaller probability. This pseudorandom wiring applies to both the neuron's data inputs as well as the neuron's control inputs, which come from the control outputs of other neurons. In SOLAR, each neuron has limited number of interconnections independent of the network size.

Hardware self-organization is a result of each neuron selecting its inputs and modifying its transformation function, to find out statistically the most efficient way of differentiating inputs from various classes. Information index is calculated based on statistics and neuron's organization is recorded as the training result.

When the system is under test, a selected control signal decides whether or not to activate a neuron to vote on an individual test vector. Statistically weighted votes of many neurons are combined to determine the final identification.

III. Single Neuron Behavior

SOLAR neurons are event-driven processors responding to their selected data and control inputs. Each neuron N receives an input vector x_i from its routing channel and transforms it to a scalar output x_o. In addition, each neuron has a binary control input τ_i and two control outputs τ_o and $\bar{\tau}_o$, as illustrated in Fig. 2.

A logic high signal in the control input activates this neuron. Control outputs are generated and statistical information is obtained when the neuron is activated. With control input high, this neuron only reacts to data from a selected part of the whole input space, which forms a local input space of this neuron. Thus, a control input plays the role of inhibitory connection in biological neurons, preventing a controlled neuron from firing. First column neurons have their control inputs set to 1, so they are always activated. Each neuron performs a simple operation like adding, subtracting and shifting, or a simple approximation of the logarithm and exponent functions. Thus, a neuron $N_{k,l}$ performs transformation from n-dimensional input space $S_{k,l}$ to one dimensional (1-D) output space S_o. Input space $S_{k,l}$ is a subspace of neuron's routing channel space

$$\Phi_{k,l} : S_{k,l} \rightarrow S_o. \tag{1}$$

Concatenation of different basic operations, which results from signal processing by several neurons, yields more complicated expressions

$$\begin{aligned} S_o &= \Phi_{k,l}(S_{k,l}) = \Phi_{k,l} \circ \Phi_{k,l-1}(S_{k,l-1}) \\ &= \Phi_{k,l} \circ \Phi_{k,l-1} \ldots \Phi_{k,1}(S_{k,1}) \equiv \Psi_{k,l}(S_{k,1}). \end{aligned} \tag{2}$$

A neuron generates partition of its input space by comparing its output against a set threshold t. The separation boundaries are defined by

$$\Psi_{k,l}(S_{k,1}) = t \tag{3}$$

which generates complex $n-1$-dimensional manifolds used to separate different classes in n-dimensional input feature space. Complexity of the resulting separation boundaries grows along with the increasing number of neuron's layers.

As a result of threshold induced separation, the neuron's input space is divided into two subspaces

$$S_{ot} = \{x \in S_{k,l} : \Phi_{k,l}(x) > t\}$$

and

$$S_{oti} = \{x \in S_{k,l} : \Phi_{k,l}(x) \leq t\}. \tag{4}$$

These subspaces correspond to two subspaces of the signal inputs space T_k and T_{ki} such that

$$S_{ot} = \Psi_{k,l}(T_k) \quad \text{and} \quad S_{oti} = \Psi_{k,l}(T_{ki}). \tag{5}$$

Each neuron is activated only when its control input is high. A neuron computes its control outputs as

$$\tau_o = \tau_i \wedge \tau_n, \quad \text{and} \quad \bar{\tau}_o = \tau_i \wedge \bar{\tau}_n. \tag{6}$$

where τ_n is a neuron's threshold function value

$$\tau_n = (\Phi_{k,l}(x) > t) \tag{7}$$

and τ_i is either τ_o or $\bar{\tau}_o$ of the control feeding neuron. A control feeding neuron is the one whose control output (τ_o or $\bar{\tau}_o$) is used as a control input to a given neuron. So, a control input is obtained by a product of neuron's threshold function values along the chain of control feeding neurons (feeding chain)

$$\tau_i = \prod_{k=1}^{m} \tau_k \tag{8}$$

where m is a number of control feeding neurons and τ_k is either τ_n or $\bar{\tau}_n$ depending on which control output was used in each feeding neuron.

According to (6) and (8), if a neuron's control output is high, all the corresponding threshold function values in the feeding chain are high. The resulting subspace of the input space is an intersection of subspaces T_k or T_{ki} for all neurons in the feeding chain.

Thus, when a neuron is activated we obtain two threshold defined subspaces

$$S = \left(\bigcap_{k=1}^{m} \hat{T}_k\right) \cap T_k \text{ and } S_i = \left(\bigcap_{k=1}^{m} \hat{T}_k\right) \cap T_{ki} \tag{9}$$

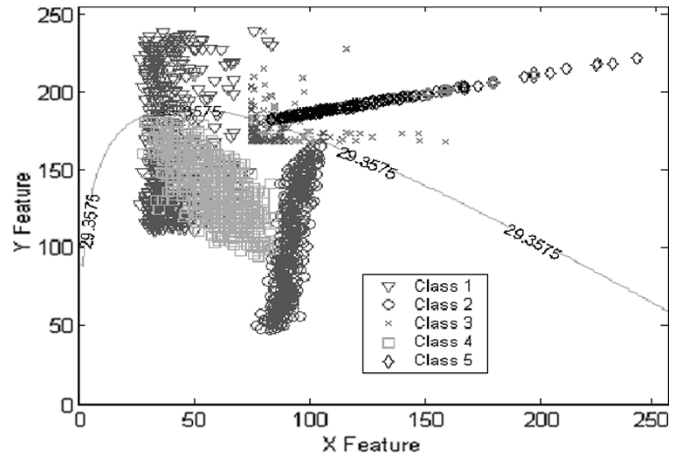


Fig. 3. Concatenation of multiple functions.

where

$$\hat{T}_k = \begin{cases} T_k & if\ \tau_k = \tau_n \\ T_{ki} & if\ \tau_k = \bar{\tau}_n \end{cases}. \tag{10}$$

Classification performed by each neuron is based on estimated class probabilities in subspaces S and S_i as defined in Section IV. Neuron's transformation functions and thresholds are optimized by contributing neurons to perform better division of classes using training data.

Example: Fig. 3 serves as an illustration for this process. It contains synthetically generated data from five classes. The following equation obtained by a second column neuron illustrates a learned combination of simple functions that results in a more complex function and provides data separation as illustrated by the separating line on Fig. 3:

$$\Psi_{k,l} = \frac{Y}{2} - \left[\frac{\log(X) - X}{2}\right] = 29.3575 \tag{11}$$

where X and Y are two features from the original input space, and 29.3575 is the equivalent threshold value in the input space. It has to be noted that the "log" function used in (11) is a rescaled base-2 logarithm, which has the same input and output range. It has been defined in this way for convenience of hardware implementation. The obtained partition is optimized to separate five classes into two subspaces, and the transformation is generated by a neuron close to the original data input.

A clear cut with a proper threshold provides a strong support for final classification. Since each neuron separates its input data into two subspaces, the quality of this separation is evaluated based on statistical measures as discussed in Section IV. Searching for the optimum threshold, as well as selecting a neuron's operation, is a training task.

IV. Neuron Self-Organizing

Self-organization of each neuron is a result of learning based on the system entropy. In training, each neuron randomly selects a number of inputs (or a single input) from the routing channel, and computes its output by performing a transformation of its inputs. The output data is then compared to predefined threshold values and the input sample is assigned to one of the threshold defined subspaces, S or S_i (9).

Using the probabilities of training data from different classes that fall into each subspace defines the information index obtained from (12) shown at the bottom of the page, where P_{sc} is probability of a class c satisfying threshold, P_{sic} is probability of a class c not satisfying threshold, P_s is subspace probability (percentage of input data that passes threshold), P_{si} is complementary subspace probability (data that does not pass threshold), and P_c is class probability.

Different combinations of data inputs, transformation operations and control inputs result in different information index values. Neuron calculates information index for different inputs and the optimized information index is stored together with neuron's configuration data, class probabilities, threshold values, etc. The information index defined by (12) is normalized to (0,1) interval. When $\mathrm{I} = 0$, there was no reduction in data entropy, while $\mathrm{I} = 1$ indicates that data entropy in neurons' input space was reduced to 0. The value of I measures the quality of this neuron's subspace separation. It is related to the definition of information deficiency introduced in (13), which quantifies the amount of information left in the subspaces.

The neuron's parameters, threshold and connectivity are a result of local optimization performed on the information index. This process is explained in the following on the example of threshold selection.

Iterative Threshold Selection: Finding the optimal threshold that yields the largest information index from (12) requires estimation of probability density functions of all classes in all subspaces, which may be a very time-consuming process. To simplify threshold evaluation, we use an iterative search by a process similar to binary search algorithm. As mentioned, a threshold separates the input space into two subspaces. The number of data points within a threshold divided subspace provides the estimated subspace probability P_s. It is convenient to find a threshold t_2 that satisfies $P_s = 0.5$. This threshold t_2 divides the input data into two groups with equal number of data

$$\begin{aligned} \mathrm{I} &= 1 - \frac{\Delta \mathrm{E}_s}{\mathrm{E}_{\max}} \\ &= 1 - \frac{\left[\sum\limits_{sc} P_{sc}\log(P_{sc}) - P_s\log(P_s)\right] + \left[\sum\limits_{sic} P_{sic}\log(P_{sic}) - P_{si}\log(P_{si})\right]}{\sum\limits_{c} P_c\log(P_c)} \end{aligned} \tag{12}$$

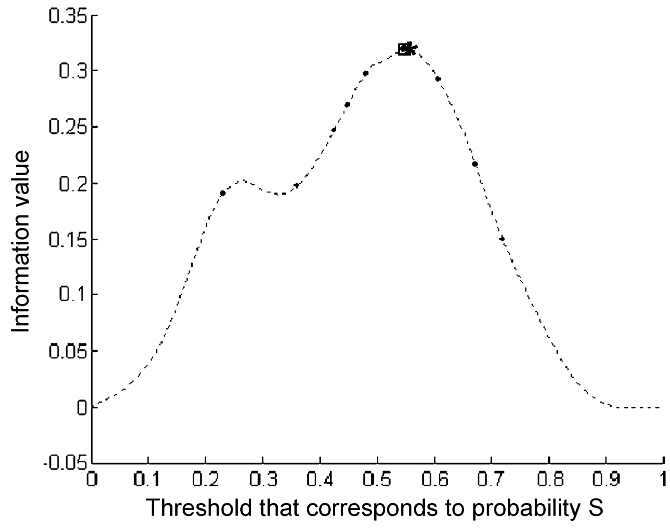


Fig. 4. Information index and its approximation.

points. Likewise, threshold t_1 and t_3 can be found from each of these two groups, respectively, which cut the two subspaces into half again. As a result, t_1, t_2, and t_3 are corresponding to subspace probability $\mathbf{P_s} = \{0.25, 0.5, 0.75\}$, respectively. Iteratively, threshold values corresponding to dichotomizing P_s values can all be determined. Based on the computed information indexes at dichotomizing threshold values, the optimum threshold searching algorithm can be organized as follows.

1) Compute threshold values $\{t_1, t_2, t_3\}$ at subspace probabilities equal to $\mathbf{P_s} = \{0.25, 0.5, 0.75\}$ and the corresponding information index values $\{\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3\}$. Find $\mathrm{I}_{\max 0} = \max\{\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3\}$, and choose P_s and t that corresponds to the maximum index value.
2) Set probability step $\Delta P_s = 0.125$ to obtain the next level of dichotomizing P_s values $\mathbf{P_s} = \{P_s - \Delta P_s, P_s, P_s + \Delta P_s\}$. Set small information increment ε and $\mathrm{i} = 0$.
3) Compute $\{\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3\}$ at subspace probabilities $\mathbf{P_s} = \{P_s - \Delta P_s, P_s, P_s + \Delta P_s\}$, and set $\mathrm{i} = \mathrm{i} + 1$.
4) Find $\mathrm{I}_{\max \mathrm{i}} = \max\{\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3\}$ and record the corresponding P_s and the threshold value. Set $\Delta P_s = \Delta P_s/2$, and compute $\Delta \mathrm{I} = \mathrm{I}_{\max \mathrm{i}} - \mathrm{I}_{\max(\mathrm{i}-1)}$
5) If $\Delta \mathrm{I} > \varepsilon$, go to 3 else stop.

This algorithm results in a fast convergence to the maximum of the information index, and is resistant to local maxima. The results of this algorithm obtained in a six class problem with 10000 samples are illustrated on Fig. 4. In this figure, square indicates the final information at the selected threshold, while star indicates the threshold value of the maximum information. For reference, Fig. 4 shows the exact values of information index (dashed line) and its discrete values computed in the search algorithm. In this case, nine evaluations were necessary to reach for the maximum with $\varepsilon = 0.02$.

Each neuron actively searches its interconnections and adopts its threshold and transformation function to attain the maximum information index.

Information Deficiency: Related to information index is information deficiency. Information deficiency is simply a normalized relative entropy value in a local subspace and is defined as follows:

$$\delta_s = \frac{\Delta \mathrm{E}_s}{\mathrm{E}\max} = \frac{\sum\limits_{sc} P_{sc}\log(P_{sc}) - P_s\log(P_s)}{\sum\limits_{c} P_c\log(P_c)}. \tag{13}$$

Information deficiency helps to self-organize the learning process. A subspace with zero information deficiency does not require any learning—data is well classified in the subspace. An important role of information deficiency is that it can be used as a measure of learning by a group of neurons that work on the same input data.

At the first layer of neurons, it is assumed that the input information deficiency is one. Information index is complimentary to the summation of all the subspace information deficiencies

$$1 - \mathrm{I} = \sum_s \delta_s. \tag{14}$$

The information deficiencies for output subspaces $\overline{\delta}_{OS}$ are defined as the product of local subspace information deficiencies and input information deficiencies

$$\overline{\delta}_{os} = \delta_i \delta_s \tag{15}$$

where δ_i is the input information deficiency. They become input information deficiencies δ_i of the connecting neurons.

δ_i allows a neuron to know if its input subspace has been sufficiently learned. If δ_i is less than or equal to the chosen information deficiency threshold, it indicates that not much information can be gained by further learning.

As subsequent neurons extract information from the input data, there is less and less independent information left in the data. The learning array grows by adding more neurons until the information deficiency in the subsequent neurons fall below a set threshold value.

V. Classification

Data to be classified is sent to the SOLAR network, and the network performs classification based on its training results. As a result of training, neurons internally save the correct recognition probabilities of all the classes for both of the output spaces as two probability vectors $[P_{sc}]$ and $[P_{sic}]$, where c stands for different classes. If an input data point falls in a voting neuron's input subspace, it is going to vote for this data using its estimated probabilities P_{sc} or P_{sic} as probabilities of correct classification P_{cc} for that class. The voting mechanism gathers all the information and classifies the input signal using a weight function designed after maximum ratio combination (MRC) technique used in mobile communication [25]

$$B_c = 1 - \frac{1}{1 + \sqrt{\sum\limits_{i=1}^{n}\left(\frac{1}{\frac{1}{P_{cc_i}} - 1 + \varepsilon}\right)^2}} \tag{16}$$

TABLE I
CLASS PROBABILITIES IN NEURONS' OUTPUT SPACES

	Voting Neuron Number					
	1	2	3	4	5	Winner-takes-all
Class 1	0	0.293	0.079	0.671	0.305	0.671
Class 2	0.753	0.632	0.125	0.329	0.695	0.753
Class 3	0.247	0.075	0.796	0	0	0.796

TABLE II
VOTING WEIGHT B_C FOR DIFFERENT CLASSES

Class 1	Class 2	Class 3
0.6800	0.8075	0.7960

where $P_{cc_i} = \mathrm{P}_{cc}$ of each "vote" for class c
n number of voting neurons
ε small number preventing division by zero

This weight function provides a statistically robust fusion of individual neurons vote. The classifier chooses a class c with the maximum weight B_c. The Appendix justifies the weight formula used in (16) based on the MRC principle.

Example: Assume that there are five neurons in a classification problem with three classes, and that the class probabilities estimated by each neuron at the learning stage are as shown in Table I (the sum of probabilities in each column is equal to 1). The largest probability value for each class is also listed, assuming winner-take-all voting scheme is used.

Using $\varepsilon = 0.001$, the weights of different classes for this particular input data are calculated and are shown in Table II.

Based on this result, SOLAR will classify this input as a sample from class 2. Notice that if the winner-takes-all approach was used, as shown in Table I, voting neuron no. 3 should make a decision and this particular input would be classified as class 3.

VI. SIMULATION RESULTS

Although the objective of this work is to develop parallel hardware organization for machine learning, SOLAR organization was first implemented and tested through software simulation. Two examples are used to illustrate the SOLAR performance. Both of them were selected from the University of California at Irvine database at ftp://ftp.ics.uci.edu/pub/machine-learning-databases/. Some of the data are missing or represented with symbols, which is inconvenient for numerical computation. In our previous work [26], we described how to transfer these data into numerical values based on training data set statistics.

In addition, since SOLAR is a self-organizing network, each network with different randomized prewiring can result in a different performance. In order to estimate the average performance of SOLAR, nine identical networks with different prewiring were generated. These nine networks were simulated in parallel, and a final majority voting was performed.

The first example is a credit card approval problem in Australia. The increasing number of credit card applications makes their manual handling very costly. Machine learning methods including ANNs [27], [28] and SOLAR can be used to facilitate this task. The data set of this example consists of 690 instances with 16 features, including class attribute, and is divided into two classes. Since the credit card data set did not separate testing data from training data, the same experimental setup as used in experiments reported in literature [15] was used, in order to compare the results. This setup used cross validation technique [15], [26] to divide the dataset randomly into n mutually exclusive data groups with equal size. During each training and testing process, one group was selected to be a testing group, while the remaining $(\mathrm{n} - 1)$ groups were used as training data. The whole procedure was repeated n times and each group was tested only once. The error rate is the average error rate of n groups. This approach eliminates statistical biases, and the error rate can be estimated efficiently. Similar to the other experiments, n was set to 10 in this simulation.

TABLE III
COMPARISON RESULT FOR CREDIT CARD APPROVAL DATASET

Algorithm	Error Rate	Algorithm	Error Rate
Cal5	0.131	Backprop	0.154
SOLAR	0.133	C4.5	0.155
Itrule	0.137	SMART	0.158
SVM$^{\text{light}}$	0.138	Baytree	0.171
BSVM	0.138	k-NN	0.181
Discrim	0.141	NewID	0.181
Logdisc	0.141	AC^2	0.181
DIPOL92	0.141	LVQ	0.197
CART	0.145	ALLOC80	0.201
RBF	0.145	CN2	0.204
CASTLE	0.148	Quadisc	0.207
NaiveBay	0.151	Default	0.440
IndCART	0.152	Kohonen	Failed

SOLARs results are compared with results from other algorithms [15], [29] as shown in Table III. In this problem, SOLAR performed fairly well among all the algorithms. Although it loses to the decision tree algorithm CAL5, it has the best performance among all artificial neuronal networks, which are highlighted in Table III. Especially, two realizations of the support vector machine (SVM) algorithm, named as SVM$^{\text{light}}$ and BSVM, have also been applied to this dataset [29]. Single neuron's operation of SOLAR is believed to have certain similarity to the behavior of SVM since the input data space is divided using threshold cutting in both methods. On this data set, SOLAR provides slightly better performance than these two SVMs.

Fig. 5 shows a randomly prewired seven-layer SOLAR network used for this data set, and Fig. 6 is the network after learning.

The second example is an adult income-based loan decision problem. There are two sets of data, a training set, which has 32561 instances and a testing set, which has 16281 instances. Both have 15 features including class attribute, and they are also divided into two classes. This data set also contains both symbolic values and missing data. Classification results of SOLAR and other methods are compared in Table IV.

Although SOLAR does not perform as well as the best algorithms for the adult income problem, it is the only learning

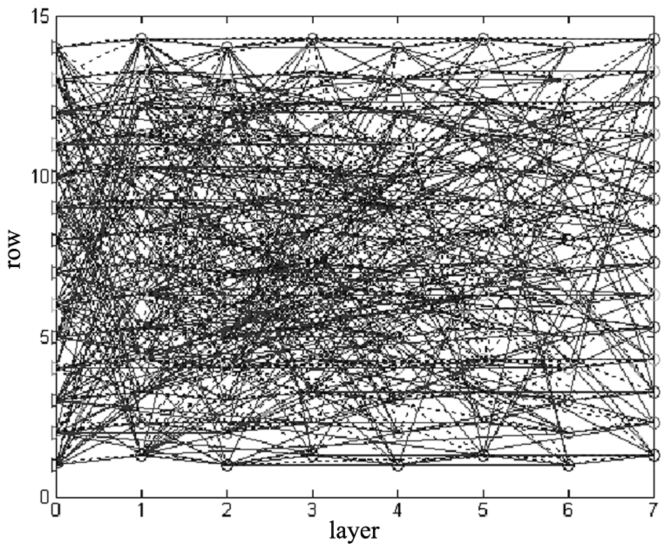


Fig. 5. Prewired network of SOLAR.

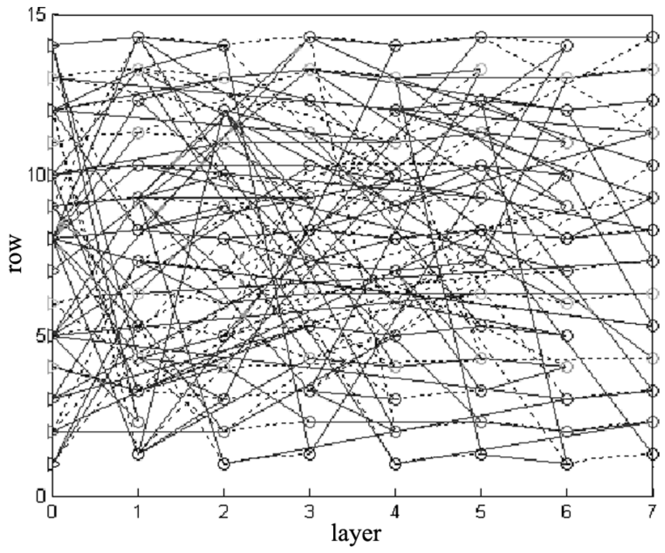


Fig. 6. Learned network of SOLAR.

neural network on the list. It should be noted that SOLAR was not designed for any specific classification or recognition tasks while decision trees tend to perform better in this type of problem [15].

VII. Conclusion and Future Work

SOLAR is a new hardware oriented machine-learning concept aimed at large sparsely connected structure for brain like computing applications. It has new biologically inspired network organization based on a fixed lattice of distributed, parallel processing units (neurons), which is easily expandable to a large system. It organizes itself into a multilayer structure, and its size is automatically self-adjusting to the complexity of the target problems. The evolution of its structures is data driven. It evolves its structures and makes recognition decisions based on statistical learning and minimum local subspace entropies. Not only this is faster than back propagation NN in training, it is much easier to implement in hardware for online learning and adaptation. The reconfigurable processing units and self-organizing structure make SOLAR adaptive to diverse recognition tasks. Interconnections among neurons are dynamically refined and grow linearly with the number of neurons. The system is easily scalable, and its hardware cost grows proportionally to the number of neurons.

TABLE IV
Comparison Result for Adult Income Dataset

Algorithm	Error Rate
FSS Naïve Bayes	0.1405
NBTree	0.1410
C4.5-auto	0.1446
IDTM (Decision table)	0.1446
HOODG / SOLAR	0.1482
C4.5 rules	0.1494
OC1	0.1504
C4.5	0.1554
Voted ID3 (0.6)	0.1564
CN2	0.1600
Naïve-Bayes	0.1612
Voted ID3 (0.8)	0.1647
T2	0.1687
1R	0.1954
Nearest-neighbor (3)	0.2035
Nearest-neighbor (1)	0.2142
Pebls	Crashed

This paper discusses the statistical learning algorithm of a feedforward static SOLAR and demonstrates its pattern recognition performance through software simulation. Tested with two common benchmark problems, SOLAR proved to be a reliable solution for the machine learning.

Dynamic feedback SOLAR structures with programmable communication channels are fit to implement large associative memories based on sparsely connected Hebbian networks analyzed in [20]. However, detailed discussion of the hardware implementation of dynamic SOLAR architectures is beyond the scope of this paper.

Obviously, SOLAR did not address many critical aspects of self-organizing machines which are needed for them to exhibit intelligent behavior. Many such issues were pointed out in [30] and [31] and have to be considered for brain-type signal processing. One of the critical issues is reinforcement driven self-organization of the control structures, which will govern the embedded self-evaluation and build the value system for the machine. The other issue is distributed memory and association of temporal events taking place in the machine. These issues are currently under investigation.

More information on SOLAR project can be found at: http:// www.ent.ohiou.edu / ~webcad / Current_Projects / solar / index.html

Appendix
Derivation of the Weighting Formula

Weighting formula (16) determines a relative strength of belief that combines the classification evidence from independent sources of information. Formula (16) is derived using analogy to the MRC of signals from various communication channels in wireless communication. Assume that $s_1, \ldots, s_k$ represent signals received from k channels all with the same noise level $n_{io} = 1$, $i = 1, \ldots, k$. The weighted signal power is $s^2 = |\sum_{i=1}^{k} w_i s_i|^2$ and the combined noise power represented by its variance is

$$\mathrm{var}(n_0) = \sum_{i=1}^{k} \mathrm{var}(n_i) = \mathrm{var}(n_{io}) \sum_{i=1}^{k} w_i^2 \text{ with} \sum_{i=1}^{k} w_i = 1.$$

The optimum way of weighting signals received through various channels is to weight them proportionally to the signal strength. So, $w_i = |s_i| / \sum_{m=1}^{k} |s_m|$ and the signal-to-noise ratio can be obtained from

$$\frac{S}{N} = \left|\frac{s^2}{\mathrm{var}(n_0)}\right| = \frac{\left(\sum_{i=1}^{k} \frac{|s_i|^2}{\sum_{m=1}^{k} |s_m|}\right)^2}{\mathrm{var}(n_{io}) \times \sum_{i=1}^{k} \frac{|s_i|^2}{\left|\sum_{m=1}^{k} s_m\right|^2}} = \sum_{i=1}^{k} \left|\frac{s_i^2}{\mathrm{var}(n_i)}\right| \tag{17}$$

So, in the optimally weighted signal, the signal to noise ratio increases as a square root of sum of squares of individual signal to noise ratios.

We use a similar approach to fuse classification results from independent classifiers. Suppose that each of k classifiers has its own probability of correct classification equal to P_{cci}. When $P_{cci} = 1$ there is no ambiguity and a classification decision is certain. By analogy to signal quality we would assign the corresponding signal to noise ratio equal to infinity. Thus, we propose first to change probability measure P_{cci} to a new variable $\widehat{P}_i = P_{cci}/1 - P_{cci}$ that would map P_{cci} to such defined signal to noise ratio $\widehat{P}_i^2 = s_i^2/\mathrm{var}(n_i)$. Using the optimum weighting of signals with different strength based on (17) we can determine the resulting variable $\widehat{P}$ as

$$\hat{P} = \sqrt{\sum_{i=1}^{k} |\widehat{P}_i|^2} \tag{18}$$

or equally

$$\frac{1}{\frac{1}{P_{cc}} - 1} = \sqrt{\sum_{i=1}^{k} \frac{1}{\left|\frac{1}{P_{cci}} - 1\right|^2}}. \tag{19}$$

Finally, solving with respect to P_{cc}

$$P_{cc} = \frac{1}{1 + \frac{1}{\sqrt{\sum_{i=1}^{k} \frac{1}{\left|\frac{1}{P_{cci}} - 1\right|^2}}}} = 1 - \frac{1}{1 + \sqrt{\sum_{i=1}^{k} \frac{1}{\left|\frac{1}{P_{cci}} - 1\right|^2}}} \tag{20}$$

and replacing P_{cc} by B_c as well as adding small ε for numerical stability, we obtain the probability related belief expressed by (16) that the resulting classification based on fused classifier's decision is correct. Notice that this belief is always stronger than the most reliable classifier's probability of correct classification $P_{cc\,\max}$.

References

[1] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[2] R. S. Sutton and A. Barto, *Reinforcement Learning an Introduction*. Cambridge, MA: MIT Press, 1998.

[3] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[4] T. Hoya, "On the capability of accommodating new classes within probabilistic neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 450–453, Mar. 2003.

[5] F. Ham and I. Kostanic, *Principles of Neurocomputing for Science and Engineering*. New York: McGraw-Hill, 2001.

[6] A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*. New York: Wiley, 1993.

[7] G. Lendaris, "On the definition of self-organizing systems," *Proc. IEEE*, vol. 52, no. 3, pp. 324–325, Mar. 1964.

[8] T. Kohonen, *Self-Organizing Maps*. New York: Springer-Verlag, 1995.

[9] J. S. Albus, "A new approach to control: the Cerebellar Model Articulation Controller (CMAC)," *Trans. ASME J. Dyn. Syst. Meas. Control*, ser. G, no. 97, pp. 270–277, Sep. 1975.

[10] J. S. Ker, Y. H. Kuo, R. C. Wen, and B. D. Liu, "Hardware implementation of CMAC neural network with reduced storage requirement," *IEEE Trans. Neural Netw.*, vol. 8, no. 6, pp. 1545–1556, Nov. 1997.

[11] H. Lee, C. Chen, and Y. Lu, "A self-organizing HCMAC neural-network classifier," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 15–27, Jan. 2003.

[12] T. J. Procyk and E. H. Mamdani, "A linguistic self-organizing process controller," *Automatica*, vol. 15, no. 1, pp. 15–30, 1979.

[13] P. Arabie and L. J. Hubert, Eds., *Clustering and Classification*, Singapore: World Science, 1996.

[14] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Dept. Appl. Math., Harvard Univ., Cambridge, MA, 1974.

[15] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. London, U. K.: Ellis Horwood, 1994.

[16] G. Koonar, S. Areibi, and M. Moussa, "Hardware implementation of genetic algorithms for VLSI CAD design," in *Proc. 15th Int. Conf. Computer Applications Industry and Engineering*, San Diego, CA, Nov. 2002, pp. 197–200.

[17] T. Higuchi, H. Iba, and B. Manderick, "Evolvable hardware," in *Massively Parallel Artificial Intelligence*, H. Kitano and J. Hendler, Eds. Cambridge, MA: MIT Press, 1994, pp. 339–420.

[18] B. M. Wilamowski, J. Binfet, and M. O. Kaynak, "VLSI implementation of neural network," *Int. J. Neural Syst.*, vol. 10, no. 3, pp. 191–197, Jun. 2000.

[19] A. Bermak and D. Martinez, "A compact 3-D VLSI classifier using bagging threshold network ensembles," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1097–1109, Sep. 2003.

[20] D. D. Vogel and W. Boos, "Sparsely connected Hebbian networks with strikingly large storage capacities," *Neural Netw.*, vol. 10, no. 4, pp. 671–682, Jun. 1997.

[21] J. E. Dayhoff, *Neural Network Architectures: An Introduction*. New York: Van Nostrand, 1990.

[22] M. M. Gupta, L. Jin, and N. Homma, *Static and Dynamic Neural Networks, From Fundamentals to Advanced Theory: From Fundamentals to Advanced Theory*. New York: Wiley, 2003.

[23] Virtex Datasheet, Xilinx Inc..*http://direct.xilinx.com/bvdocs/publications/ds003.pdf* [Online]

[24] J. E. Dowling, *Creating Mind: How the Brain Works*. New York: Norton, 1998.

[25] J. G. Proakis, *Digital Communications*, 3rd ed. New York: McGraw-Hill, 1995.

[26] J. A. Starzyk and T.-H. Liu, "Design of a self-organizing learning array system," in *Proc. IEEE Int. Symp. Circuits Systems*, Bangkok, Thailand, May 2003, pp. 801–805.

[27] V. S. Desai, J. N. Crook, and G. Overstreet, "A comparison of a neural network and classical techniques for credit scoring," *Eur. J. Oper. Res.*, vol. 95, no. 1, pp. 24–36, Nov. 1996.

[28] B. Baesens, R. Setiono, C. Mues, and J. Vanthienen, "Using neural network rule extraction and decision tables for credit-risk evaluation," *Manage. Sci.*, vol. 49, no. 3, pp. 312–329, Mar. 2003.

[29] C. Hsu and C. Lin, “A simple decomposition method for support vector machines,” *Mach. Learn.*, vol. 46, pp. 291–314, 2002.

[30] E. Korner and G. Matsumoto, “Cortical architecture and self-referential control for brain-like computation,” *IEEE Eng. Med. Biol. Mag.*, vol. 21, no. 5, pp. 121–133, Sep. 2002.

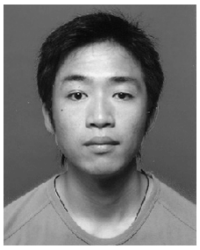
[31] J. G. Taylor, “Toward the networks of the brain: from brain imaging to consciousness,” *Neural Netw.*, vol. 12, no. 7–8, pp. 943–959, Oct. 1999.

Janusz A. Starzyk (SM’83) received the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering from Warsaw University of Technology, Warsaw, Poland, in 1971 and 1976, respectively.

From 1977 to 1981, he was an Assistant Professor at the Institute of Electronics Fundamentals, Warsaw University of Technology. From 1981 to 1983, he was a Postdoctorate Fellow and Research Engineer at McMaster University, Hamilton, ON, Canada. In 1983, he joined the Department of Electrical and Computer Engineering, Ohio University, Athens, where he is currently a Professor of electrical engineering and computer science. He has cooperated with the National Institute of Standards and Technology in the area of testing and mixed signal fault diagnosis. He has been a consultant to AT&T Bell Laboratories, Sarnoff Research, Sverdrup Technology, Magnolia Broadband, and Magnetek Corporation. His current research is in the areas of self-organizing learning machines, neural networks, rough sets, VLSI design and test of mixed signal CMOS circuits, and reconfigurable design for wireless communication.

Zhen Zhu received the B.S. degree in information and communication engineering from Xi’an Jiaotong University, Xi’an, China, in 2000 and the M.S. degree in electrical engineering from Ohio University, Athens, in 2002. He is currently pursuing the Ph.D. degree at Ohio University.

He has been working as a Research Associate at the Avionic Engineering Center, Ohio University, since 2001. He also worked for FreeFlight Systems, Waco, TX, in 2004. His research interests include machine learning systems, Global Positioning System receiver design, and signal processing.



Tsun-Ho Liu received the B.S. and M.S. degrees in electrical engineering from Ohio University, Athens, in 2000 and 2002, respectively.

In 2003, he joined the Design Center, Fujitsu Microelectronics Pacific Asia, Hong Kong, where he is responsible for customized embedded flash MCU design and new embedded flash MCU architecture development.